Ver. 4.5対応

# QuickBASIC ハンドブック vol.2 [リファレンス]

handy reference



阿部友計＝著

ナツメ社




ナツメ｜ハンディ・リファレンス……38

natsume handy reference｜38

# 目次より




イラストレーション 尾谷美幸
カバー・デザイン 早瀬芳文

# QuickBASIC ハンドブック vol.2 ［リファレンス］

阿部友計＝著

ナツメ社

ナツメ・ハンディ・リファレンス……38

handy reference

natsume handy reference 38

# はじめに

この本『Quick BASIC ハンドブック Vol.2　リファレンス』は、マイクロソフト社の最新のプログラミング言語"Quick BASIC バージョン 4.5"の使い方について書いてあります。

QuickBASIC を使えるようになるには、

- ▶統合環境
- ▶言語体系
- ▶構造化プログラミングの仕方

についてマスターしなければなりません。その環境やツールの使い方を理解しておかないと快適で効率よいプログラミングを行なえないでしょうし、またプログラムを構造化したスタイルで一貫して書かないと宝の持ち腐れになってしまいます。

この本では、

**「QuickBASIC の言語体系」**

を中心にまとめました。

**「QuickBASIC の統合環境」**

については、本書とカップルになっている同シリーズの『Quick BASIC ハンドブック Vol.1　基本操作』において扱っています。

なお、QuickBASIC を使った「構造化プログラミングの仕方」については、ハンディ・シリーズの性質上、意識して取り扱いませんでした。その代わりに、この本では、以下に掲げる

- 言語仕様
- すべてのステートメント
- すべての関数
- すべてのメタ・コマンド

について網羅的でかつ簡潔に書いてあります。また、

- はじめて使うときに行なうセットアップの仕方
- QuickBASIC の起動の仕方と終わり方
- 統合環境ウィンドウ
- メニューにおけるすべてのコマンド
- すべてのダイアログ・ボックス
- すべてのキー操作とマウス操作
- ファイル管理・編集・実行
- コマンド・ラインでの使い方
- CodeView の使い方

については、『QuickBASIC ハンドブック Vol.1　基本操作』を利用してください。

”QuickBASIC”とこの本が多くの読者に役立つことを確信しています。この本を読んで QuickBASIC の仲間が増えれば、このうえない喜びとするところです。

最後になりましたが、この本の企画および編集にあたられたナツメ出版企画㈱の山路和彦氏に感謝するとともにお礼を申しあげます。

1990 年 6 月 1 日

阿部　友計

# この本の構成

この本は、全部で5つの章から構成されています。

**《第1章　構造化された言語体系》**

QuickBASICの特徴である「構造化された言語体系」について要約してあります。

**《第2章　言語仕様》**

QuickBASICのプログラムを書く上での約束事、定数と変数、式と演算子、構造化などについて書いてあります。

**《第3章　ステートメント》**

すべてのステートメントをアルファベット順にまとめてあります。

**《第4章　関数》**

すべての関数をアルファベット順にまとめてあります。

**《第5章　メタ・コマンド》**

すべてのメタ・コマンドをアルファベット順にまとめてあります。

**《付録》**

1　ASCIIコード
2　キー・スキャン・コード
3　エラー・メッセージ
4　機能別ステートメント・関数索引
5　Ver.4.5で追加されたステートメントおよび関数

# この本の利用の仕方

**■必要なときにすぐ引ける！**

このハンディ・シリーズの本は、一度読んで終わりにする本でなく、パソコンや読者のそばに置いておいて、必要なときにサッ！　と、開いて使えるように工夫されています。したがって、この本の各章は、完全に独立したテーマについて書いてありますので、開いたところだけを読んでも理解できるようになっています。

**■初心者の人は...**

もし読者がQuickBASICについては、全くのビギナーであるならば、『**Quick BASIC ハンドブック Vol.1　基本操作**』と『**Quick BASIC ハンドブック Vol.2　リファレンス**』の両方を、ひととおり眼を通されることをおすすめします。

**■パワー・プログラマーの人は...**

また、もし読者がBASICや他の言語でプログラミングしたことがあるパワー・プログラマーであるならば、本書を重点的に読まれることをおすすめします。

さらに、プログラミング中にステートメントや関数などの書式や意味について不明な点がでてきたら本書の第3章から第5章の解説を読んでください。

また、QuickBASICの統合環境下で使っていて画面やキー操作について迷ったら『**QuickBASIC ハンドブック Vol.1　基本操作**』の該当ページを開いて読んでください。

# 目　次

## 第 1 章　構造化された言語体系



## 第 2 章　言語仕様



## 第 3 章　ステートメント

## 第 4 章 関数

## 第 5 章 メタ・コマンド

# Quick BASIC ハンドブック Vol.1　基本操作◉目　次

# [第1章]
# 構造化された言語体系

この章では、QuickBASICを
“言語体系の構造化”の観点から説明します。

# 構造化された言語体系

マイクロソフト社の「Quick BASIC」は、いままでにない新しいタイプのBASICです。これまでBASICといえば、パソコンのビギナーが遊び半分でさわる言語とか、素人が使う言語とか、満足のいくシステムを開発できない言語とか言われてきました。残念ながら正直なところ、イメージがあまり良くないようでした。

この本で取り上げているQuick BASICは、いま人気のあるC言語やPASCALと比較してもまったく遜色がありません。むしろ、ひとつの開発言語としてとらえたならば優れているところがたくさんあります。Quick BASICは、プログラムを開発するのに必要なステートメントや関数を十分なほど豊富に提供してくれています。

以下、この章では、Quick BASICの二大特徴のうちの

**構造化された言語体系**

について説明します。

「統合化された開発環境」については、『Quick BASICハンドブックVol.1　基本操作』をお読みください。

Quick BASICのプログラミングにおいて、構造化できるものには、次の4つがあります。

**▶プログラム**

プログラムを構成するモジュールやプロシージャを使って階層的で独立化したブロック・ルーチンをつくることができます。

**▶フロー**

プログラムの複雑な流れを論理的に構造化して明確にできるようになりました。

▶ **データ**

データのタイプをユーザーが定義でき、また構造的にデータ・タイプを定義することにより複雑なレコード・タイプのデータをハンドリングできるようになりました。

▶ **スコープ**

変数の有効範囲を特定のモジュール・レベル・コードまたはプロシージャ内だけで有効なローカル変数とそれらに共通して有効なグローバル変数のどちらにでも宣言できます。

## プログラムの構造化

QuickBASICでは、プログラムを構成するサブ・レベルのブロックとして、「モジュール」と「プロシージャ」が用意されています。

▶ **モジュール化**

ソース・プログラムを大きな1つのファイルにしてしまうのではなく、より細かくブレークダウンした部分に分割できることにより、管理しやすくなります。

▶ **SUB プロシージャのサポート**

モジュールの中にどこからでも呼びだして（コールして）使えるSUBプロシージャを作ることができます。QuickBASICのステートメントを標準として組み込まれているステートメントとするならば、このSUBプロシージャは、ユーザー自身が定義できるステートメントということになります。

▶ **FUNCTION プロシージャのサポート**

式の中で使えるFUNCTIONプロシージャを作ることができます。QuickBASICの関数を標準として組み込まれている関数とするならば、このFUNCTIONプロシージャは、ユーザー自身が定義できる関数ということになります。

▶ **他言語のモジュールのコール**

必要ならば、C言語・PASCAL・FORTRAN・アセンブラで開発した関数や手続きを呼びだして使えます。

より詳しくは、「**2・4　プログラムの構造化**」を参照してください。

## フローの構造化

Quick BASICでは、順次処理、分岐処理、および反復処理を使って構造化プログラミングできるように、フローを制御するためのステートメントが用意されています。

▶ **行番号なしの順次処理**

プログラムは、流れを変えないかぎり、プログラムの上から下へ順次実行されます。

▶ **ブロック IFによる複数行化した多重分岐処理**

IF...THEN...ELSEによる分岐で、真偽どちらの場合でも複数のステートメントを書けるので、構造化プログラミングを行ないやすい。またELSEIFもあるので多重に分岐した処理を行なえます。

▶ **SELECT CASEによる多重分岐処理**

複雑な分岐条件を処理できるようにSELECT CASEステートメントが用意されています。

▶ **DO LOOPによる反復処理**

前処理も後処理もできるDO...LOOPステートメントが用意されている。もちろん、FOR...NEXTやWHILE...WENDも使えます。

より詳しくは、「**2・3　フローの構造化**」を参照してください。

# データの構造化

Quick BASICでは、複数のデータを1つのまとまったデータとして取り扱えるようにする目的で、配列変数とユーザー定義変数が使えます。

### ▶範囲指定できる配列変数

同一タイプの複数の変数をまとめて処理するための変数として配列変数が使えます。単純変数およびユーザー定義型変数の配列をつくることができます。

また、配列の添字の下限と上限を自由に設定できるので、現実の世界にあったデータ処理を行なえます。

### ▶ユーザー定義型変数

異なった複数のタイプの変数を1つにまとめて処理するための変数としてユーザー定義型変数が使えます。この変数は、レコード変数ともいいます。基本となるデータ・タイプは、文字列型と数値型の2つしかないが、これら2タイプを使って定義したりすれば、様々な型を定義できます。

また、ユーザー定義型の中で他のユーザー定義型も使えるので、構造的にデータ・タイプを定義することにより複雑なレコード・データの型を定義できます。

より詳しくは、「**2・2　データの構造化**」を参照してください。

## スコープの構造化

変数は、その有効範囲をローカルなものあるいはグローバルなものとして宣言できます。

### ▶ローカル変数

特定のモジュール・レベル・コードまたはプロシージャ内だけで有効な変数としてローカル変数を宣言できます。

### ▶グローバル変数

モジュール・レベル・コードまたはプロシージャ内に共通して有効な変数としてグローバル変数を宣言できます。

## その他の特徴

その他にも、言語仕様上、サポートされている数多くの特徴があります。

### ▶インクルード・ファイル

ソース・ファイルの一部として、ソース・モジュールから独立したインクルード・ファイルが使えるようになっています。

### ▶メタ・コマンド

ステートメントおよび関数以外に、プログラムの取り扱い方や配列の記憶領域を指示するための命令としてメタ・コマンドがあります。

### ▶変数の有効範囲の宣言

モジュール間、モジュールとプロシージャ間での変数の有効範囲を、グローバルにもローカルにも宣言できます。

### ▶固定長文字列変数

文字列のタイプとして、最大32767文字までの固定長文字列を宣言できます。

### ▶長整数型数値

より桁数の多い整数を取り扱えるように、長整数型数値(long integer)がサポートされています。

### ▶ IEEE フォーマット

浮動小数点（実数）は、特に指示しない限り、MBF よりさらに精度の高い IEEE フォーマットで内部的に処理されます。

### ▶バイナリー・ファイル

シーケンシャル・ファイルやランダムアクセス・ファイルの他に、バイナリー・ファイルの読み書きが自由にできます。

### ▶再帰（recursion）

SUB プロシージャや FUNCTION プロシージャを再帰的に呼びだして使えます。

### ▶コマンド・ラインの引数

コマンドの引数を、統合環境下での実行時でもコマンド・ラインでの実行時でも取り込めます。

### ▶日本語処理

半角・全角の混在した文字列を処理するためにステートメントや関数が日本語バージョンではたくさん追加されています。

### ▶グラフィック

カラー、ピクセル、描画、座標処理などグラフィックに必要なものはすべてサポートしています。

### ▶割り込み

イベント・トラップや DOS のシステム・コールを簡単に行なえるようになっています。

# [第2章] 言語仕様

この章では、QuickBASICがプログラム言語として持っているその仕様を、
取り扱えるデータやその演算子、
およびプログラミングにおける構造化の観点から説明します。

2

# 2・1

# ソース・プログラム

## ステートメントと関数とメタ・コマンド

Quick BASICでは、コンピュータに対する命令は、「ステートメント(statement)」「関数(function)」「メタ・コマンド(meta-command)」を使って行なわれます。

### ■ステートメント

ステートメントとは、それ自体独立した機能を持つ命令のことをいいます。たとえば、OPEN、INPUT、PRINT などがステートメントです。全ステートメントについては、第3章でまとめてあります。

### ■関数

関数は、0個以上の引き数を渡すとある値を返してくれる機能を持っています。関数は、命令そのものではありませんが、ステートメント文の中で使います。たとえば、EOF()、LEFT$()などが関数です。全関数については、第4章でまとめてあります。

### ■メタ・コマンド

この命令は、プログラムの実行上、ソース・ファイルや変数の取り扱い方についてQuick BASIC に指示するためにあります。全部のメタ・コマンドについては、第5章でまとめてあります。

# ステートメント文と行

### ■ステートメント文

1つの命令を実行するために、ステートメントとこれに続けて必要なデータ・引数・スイッチなどを書いた文をステートメント文といいます。

たとえば、

```
OPEN "BOOKS.DAT" FOR INPUT #1
  ⋮
DO UNTIL EOF(1)
  ⋮
LOOP
```

では、OPEN・DO・LOOP などがステートメントであり、これらの各文がステートメント文です。また、EOF() がファイルの終わりかどうかを調べる関数です。

### ■行

#### ▶1行に書ける文字数

1行に書ける文字数は、256文字です。

#### ▶マルチ・ステートメント行

ステートメント文は、1行に1つのステートメント文ずつ書きますが、ステートメント文を"：(コロン)"で区切って1行に2つ以上のステートメント文を書くことができます。このように2つ以上のステートメント文を書いた行を「マルチ・ステートメント行」といいます。

たとえば、

```
LOCATE 10, 5 : COLOR 3 : PRINT "Quick BASIC"
```

のようにも書けます。特に必要がないかぎり、1行に1つのステートメント文を書いた方が慣れてくると見やすいようです。

▶ **行番号**

伝統的なBASICでは、各行には「行番号」が必ずありました。しかし、Quick BASICでは、付けても付けなくても、どちらでもステートメントを実行できます。行番号を付けられるようになっているのは、今までのBASICとの互換性のためであって、『構造化プログラミング』の観点からすれば必要ありません。もし付けるときは、

**0〜65529までの整数**

を使って、キーボードからプログラマー自身が入力して付けます。

▶ **行ラベル**

特定の行に名前（「行ラベル」）を付けることもできます。行ラベルは、40文字以内で付け、最初の1文字はアルファベットでなければならず、その最後に ：（コロン）を付けなければなりません。ただし、大文字と小文字の区別はありません。たとえば、

**DrawMainMenu**：

のように付けます。

ソース・プログラム

## コメント

ソース・ファイルの中にコメント（注釈）を書きます。

コメントを書くには、2つの方法があります。

■ **REMステートメントを使ってコメントを書く方法**

1つのステートメントとして書きます。

```
REM  プログラム名 - MAILING.BAS
REM       改訂日 - 1989.05.31
OPEN "LETTER.DAT" FOR RANDOM AS 1： REM ファイル・オープン
  ：
```

■'記号に続けてコメントを書く方法

"："で区切らなくてもステートメントの右に書けます。

```
'  プログラム名 - MAILING.BAS
'  改訂日 - 1989.05.31
OPEN "LETTER.DAT" FOR RANDOM AS 1 'ファイル・オープン
   ⋮
```

ソース・プログラム

# 2・2

# データの構造化



## データ：文字列型と数値型

Quick BASICで取り扱えるデータのタイプは、基本的には、「文字列(string)型」と「数値(numeric)型」だけです。論理型も日付型も金額型もありません。しかし、プログラマーは、文字列型と数値型を組み合わせて、独自のデータ・タイプをプログラマー自身が定義できるようになっています。

### ■文字列型

文字列型のデータは、0文字以上の長さを持つ文字です。1つの文字列の長さは、0〜32,767文字まで取り扱えます。文字長が0の文字列をヌル(null)といい、ASCIIコードのchr$(0)と同じです。

文字列型のデータには、「可変長文字列」と「固定長文字列」の2種類があります。

◉文字列型データの種類

| 種　類 | 特　徴 | 例 |
|---|---|---|
| 可変長文字列<br>variable－length string | 文字の長さが一定でない文字列です。 | "Quick BASIC"<br>"構造化"<br>"" |
| 固定長文字列<br>fixed-length string | 文字の長さを必要なだけ前もって宣言した文字列です。<br>その長さよりも短い文字列を代入したいときは、残りの部分が空白になります。 | "Quick　　"<br>"QB　　　"<br>"　　　　" |

## 数値型

数値型のデータには、「整数」・「長整数」・「単精度型実数」・「倍精度型実数」の 4 種類あり、それぞれ異なった範囲の数値を扱えます。

Quick BASIC では実数を MBF ではなく IEEE フォーマットで処理します。

データの構造化

◉数値型データの種類

| 種類とバイト数 | 範　囲 | 例 |
|---|---|---|
| 整数<br>INTEGER<br><br>2バイト | -32768 ～ +32767 (10進)<br>&O0 ～ &O177777 ( 8進)<br>&H0 ～ &HFFFF (16進) | 128，-63<br>&O177<br>&H8AC0<br>0， |
| 長整数<br>LONG integer<br><br>4バイト | -2147483648～+2147483647(10進)<br>&O0 ～ &O37777777777 ( 8進)<br>&H0 ～ &HFFFFFFFF(16進) | 150069<br>&O25550<br>&H4A71DC |
| 単精度型実数<br>SINGLE precision<br>floating point<br>4バイト | -3.402823E+38 ～ -1.175494E-38<br>+1.175494E-38 ～ +3.402823E+38 | 3.14159 |
| 倍精度型実数<br>DOUBLE precision<br>floating point<br>8バイト | -1.797693134862316D+308 ～<br>-2.225073858507201D-308<br>+2.225073858507201D-308 ～<br>+1.797693134862316D+308 | 0.3775049 |



## 定数と変数

プログラムの中でデータは、「定数(constant)」または「変数(variable)」にして取り扱います。

### ■定数

定数とは、プログラムの実行中変化することのないデータです。これには、「リテラル定数」と「シンボリック定数」の2種類があります。

●定数の種類

| 種類 | 文字列 | 数値 |
|---|---|---|
| リテラル定数<br>literal constant<br>データそのものをいう | ●""で囲まなければならない。<br>"Quick BASIC"<br>"構造化"<br>""<br>●ASCIIコードおよび漢字連続コードは、数字で表わすことができる。<br>CHR$(65)<br>CHR$(257)<br>●JISコードは、先頭に&Jを付ける。<br>CHR$(&J2122)<br>●句点コードは、先頭に&Kを付ける。<br>CHR$(&K0102) | ●整数(10進表現)は符号以外何も付けない<br>12345<br>-258<br>●整数(8進表現)は、先頭に&Oを付ける。<br>&O177<br>●整数(16進表現)は先頭に&Hを付ける。<br>&H1A<br>●長整数(10進表現)は、符号以外何も付けない。<br>534210187<br>-84900000<br>●長整数(8進表現)先頭に&Oを付ける。<br>&O177<br>●長整数(16進表現)先頭に&Hを付ける。<br>&H1A<br>●単精度型実数は、指数形式としてEで表わす<br>5.12733E+12<br>●倍精度型実数は、指数形式としてDで表わす<br>2.01234567890123D-03 |
| シンボリック定数<br>(記号定数)<br>symbolic constant 定数に名前を付けて、あとはその名前を使う | CONST title$ = "Quick"<br>⋮<br>PRINT title$ | CONST start = 1<br>CONST max = 100<br>⋮<br>FOR i = start TO max<br>⋮<br>NEXT i |

データの構造化

## ■変数

変数とは、プログラムの実行中に自由に変えることのできるデータです。

### ▶変数名

**変数の名前を「変数名」といいます。この変数名について、つぎのようなルールがあります。**

- **アルファベット文字・数字・ピリオドを使って付ける。**
- **最大40文字以内で付ける。**
- **アルファベット文字で始まる。**
- **アルファベット文字の大文字と小文字の区別はない。**
- **予約語は使えない。**

### ▶型宣言

変数名がどのタイプのデータを取り扱うかは、その型を宣言することにより決まります。その型宣言の仕方として3つあります。

◉変数の種類

| 方法＼型 | 最初の代入時または使用時に型宣言記号を変数名の末尾に付ける。<br>付けないと単精度型となる。<br>以後その記号を付けなければならない。 | 型宣言ステートメントを使う。 | ※ COMMON<br>※ DIM<br>※ REDIM<br>※ SHARED<br>※ STATIC<br>※ TYPE<br>のステートメントで<br>AS タイプ<br>により型宣言を行なう。<br>以後記号は、付けない。 |
|---|---|---|---|
| 整数 | year% = 1989 | DEFINT | AS INTEGER |
| 長整数 | amount& = 100000 | DEFLNG | AS LONG |
| 単精度型実数 | PI! = 3.14 | DEFSNG | AS SINGLE |
| 倍精度型実数 | PI # = 3.141598 | DEFDBL | AS DOUBLE |
| 可変長文字列 | aName$ = "QB" | DEFSTR | AS STRING |
| 固定長文字列 | ―――― | ―――― | AS STRING ＊n |

＊ｎは、固定長文字列の長さを指定する

＊ユーザー定義型の宣言については後述の箇所を参照のこと

これらの可変長文字列型変数・固定長文字列型変数・整数型変数・長整数型変数・単精度型変数・倍精度型変数を総称して「単純変数」といいます。

## 構造化した変数：配列とユーザー定義型

Quick BASICでは、複数の単純変数とこれから述べるユーザー定義型変数を構造化したデータとして取り扱うために、「配列変数(array variable)」と「ユーザー定義型変数(user-defined type variable)」の２つが使えるようになっています。

### ■配列変数

配列は、"同一タイプの複数の変数をまとめて処理する"のに適しています。そのための変数を配列変数といいます。単純変数およびユーザー定義型変数の配列をつくることができます。

#### ▶形式

配列変数は、つぎの形式になっています。

**配列変数名（次元1，次元2，...）**

配列変数名は、配列全体に付けた名前です。最大60次元まで確保できます。

#### ▶要素と添字

配列を構成している１つ１つのことを「要素」といい、それらを特定するための番号を「添字」といいます。

たとえば、整数型の1次元配列の宣言を

```
DIM month%(12)
```

のようにしたときは、つぎのようなイメージで考えればよいでしょう。

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 配列変数 month% | 0 | 31 | 28 | 31 | 30 | 31 | 30 | 31 | 31 | 30 | 31 | 30 | 31 |
| 添字 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |

5番目の要素は、month%(5)のように表わし、そのデータは31になっています。

### ▶下限と上限

添字には、下限と上限があります。特に指定しないと下限は0か1になっています（OPTION BASEステートメントでどちらかに設定できます）。上限は、宣言したときの数字が上限の添字になります。

けれども、必要とするならば、下限と上限の両方を指定できます。たとえば、

```
DIM year (1980 TO 1985)
           ↑       ↑
          下限     上限
```

としたときは、6要素の配列を宣言したことになります。

## ■ユーザー定義型変数

ユーザー定義型は、"異なった複数のタイプの変数を1つにまとめて処理する"のに適しています。ユーザー定義型変数は、レコード変数ともいいます。

データの構造化

▶**形式**

ユーザー定義型変数の宣言に先だって、TYPEステートメントを使って最初にその型宣言をしなければなりません。

```
TYPE  ユーザー定義型名
  要素名1  AS  タイプ
  要素名2  AS  タイプ
     ⋮
  要素名n  AS  タイプ
END  TYPE
```

ユーザー定義型名は、その型全体に付けた名前です。ユーザー定義型の宣言に含めることのできるタイプは、固定長文字列型・整数型・長整数型・単精度型・倍精度型・ユーザー定義型で、可変長文字列と配列を含めることはできません。

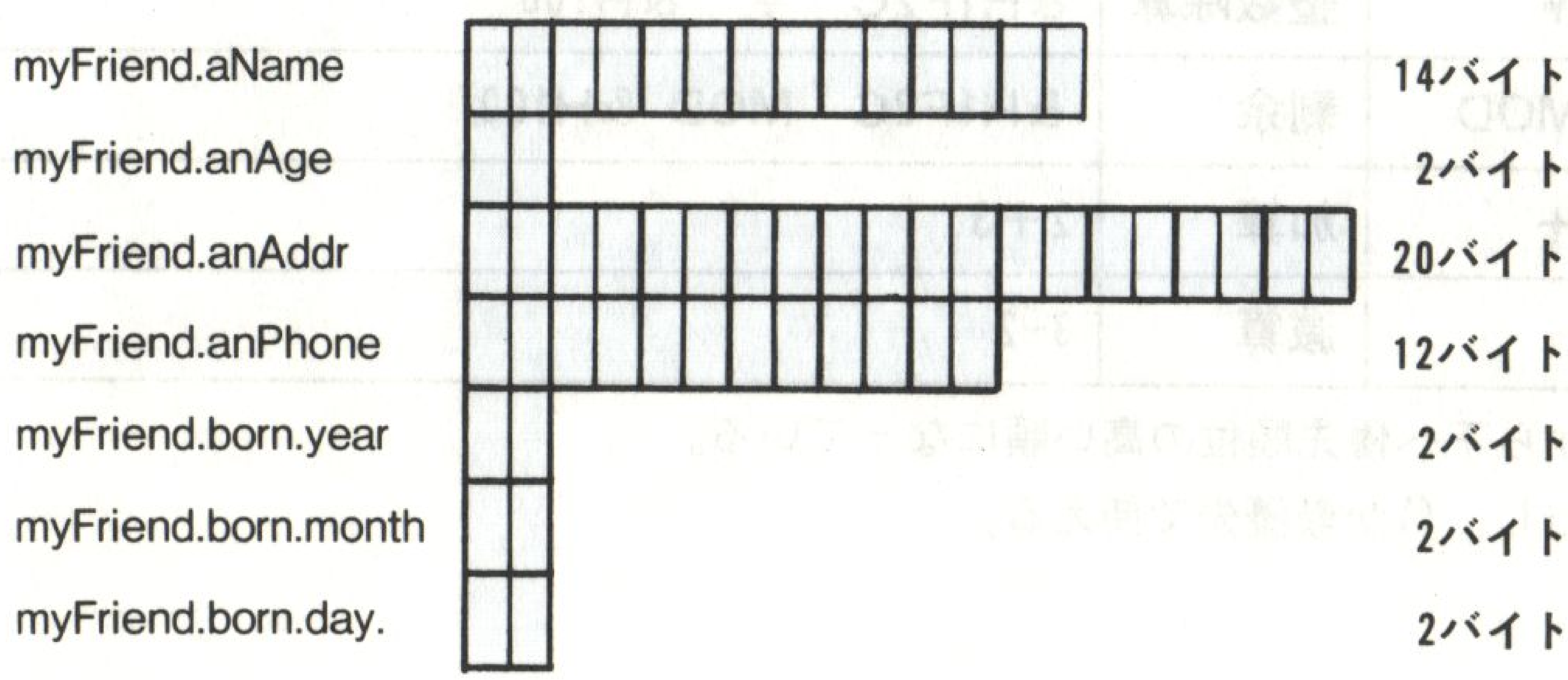
次に、そのユーザー定義型名を使って実際のユーザー定義型変数を宣言します。

```
DIM  ユーザー定義型変数名  AS  ユーザー定義型名
```

たとえば、

```
TYPE birthdayType
  year     AS INTEGER
  month    AS INTEGER
  day      AS INTEGER
END TYPE
TYPE friendInfoType
  aName     AS STRING * 14
  anAge     AS INTEGER
  anAddr    AS STRING * 20
  anPhone   AS STRING * 12
  born      AS birthdayType
END TYPE
DIM myFriend AS friendInfoType
```

とした場合、次のようなイメージで考えればよいでしょう。




※ユーザー定義型変数 myFriend は、全部で54バイトの大きさになります。実際には、パックされています。

※ユーザー定義型変数のデータは、各要素別に取り扱うこともできますし、またユーザー定義型変数名を指定するだけで同一タイプ間での代入・ファイルアクセス・プロシージャへの引数渡しなど様々なところで使えます。

## 式と演算子

データの操作は、「式」で行ないます。式とは、定数・変数および関数の各々、もしくはそれらを演算子でつなげたもののことをいいます。

演算子には、つぎの4種類があります。

### ■算術演算子

算術演算子とは、数値データを操作するための演算記号です。

◉算術演算子の種類

| 算術演算子 | 意味 | 例 | 結果 |
|---|---|---|---|
| ^ | ベキ乗 | 2^3 | 8 |
| ＊ | 乗算 | 2＊3 | 6 |
| / | 実数除算 | 15.3/5.1 | 3 |
| ¥ | 整数除算 | &H1F2C ¥ &H100 | &H1F |
| MOD | 剰余 | &H1F2C MOD &H100 | &H2C |
| ＋ | 加算 | 2＋3 | 5 |
| − | 減算 | 3−2 | 1 |

※上から下へ優先順位の高い順になっている。

※ただし、()が最優先で使える。

■文字列演算子

文字列演算子とは、文字列データを操作するための演算記号です。

◉文字列演算子の種類

| 文字列演算子 | 意味 | 例 | 結果 |
|---|---|---|---|
| + | 連結 | "Quick" + "BASIC" | QuickBASIC |

■関係演算子

関係演算子とは、同じタイプのデータを比較するための演算記号です。真のとき−1になり、偽のとき0になります。

◉関係演算子の種類

| 関係演算子 | 意味 | 例 | 結果 |
|---|---|---|---|
| = | 等しい | 10 = 9 | 0 |
| < > | 等しくない | 10 < > 9 | -1 |
| < | 未満 | 10 < 9 | 0 |
| <= | 以下 | 10 <= 9 | 0 |
| > | 超過 | 10 > 9 | -1 |
| >= | 以上 | 10 >= 9 | -1 |

※関係演算の結果は、真（-1）または偽（0）になる。

## ■論理演算子

論理演算子とは、複数の関係演算の論理を求めるための演算記号です。

◉論理演算子の種類

| 論理演算子 | 意味 | 例 | 結果 |
|---|---|---|---|
| NOT | 否定 | male$ = "Y" : female$ = "N"<br>NOT male$ = "N" | −1 |
| AND | 論理積 | male$ = "Y" : female$ = "Y"<br>male$ = "Y" AND female$ = "Y" | −1 |
| OR | 論理和 | male$ = "Y" : female$ = "N"<br>male$ = "Y" OR female$ = "Y" | −1 |
| XOR | 排他的論理和 | male$ = "Y" : female$ = "N"<br>male$ = "Y" XOR female$ = "Y" | −1 |
| EQV | 同値 | male$ = "N" : female$ = "N"<br>male$ = "Y" EQV female$ = "Y" | −1 |
| IMP | 包含 | male$ = "Y" : female$ = "Y"<br>male$ = "N" IMP female$ = "Y" | −1 |

論理演算の結果は、真（-1）または偽（0）になる。

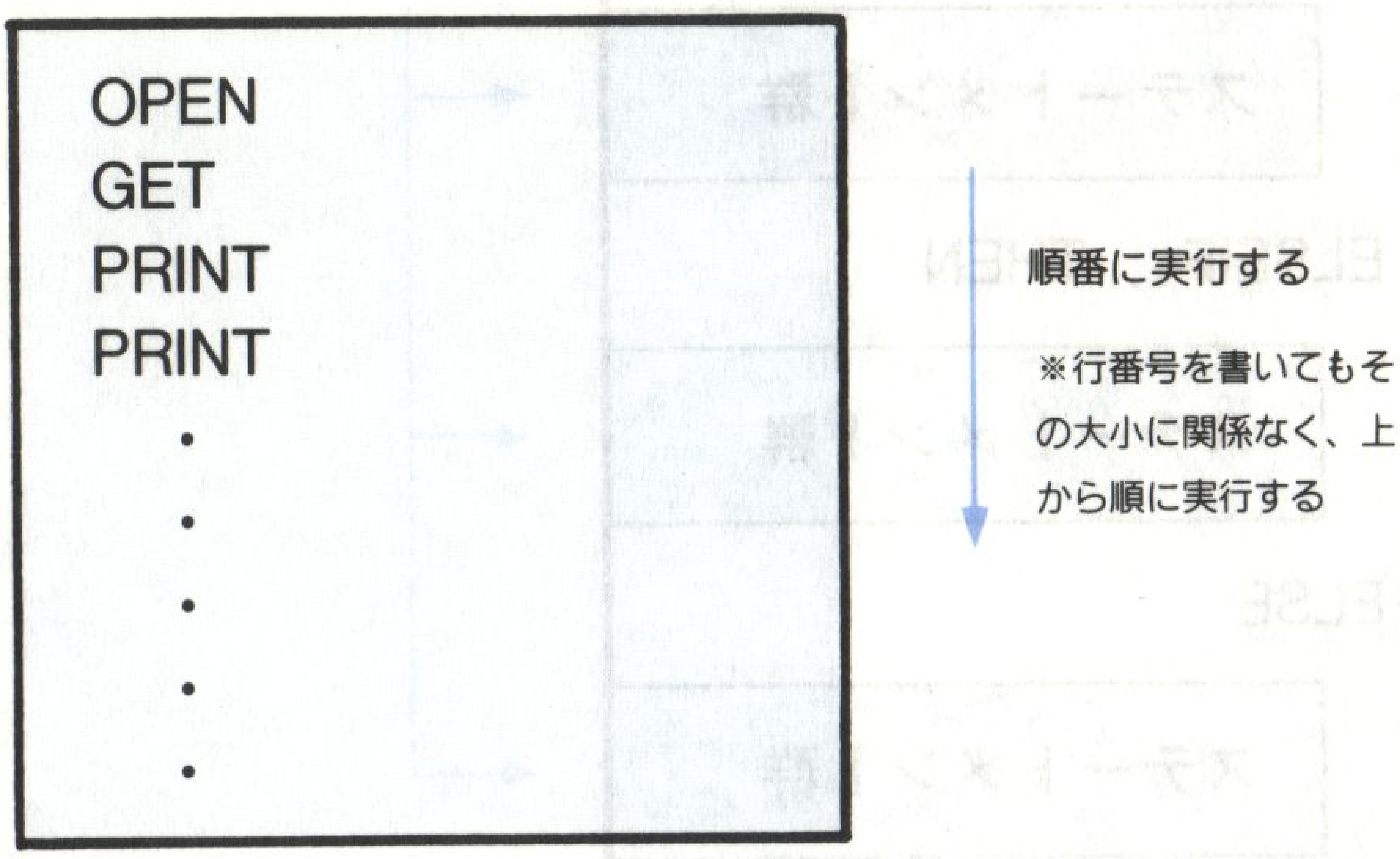
# 2・3

# フローの構造化

構造化プログラミングのアルゴリズムは、3つの基本構造を組み合わせて構築します。

### ■順次構造

Quick BASICのプログラムは、行番号の有無に関係なくソース・ファイルの上の行から下の行へステートメントを実行します。マルチ・ステートメントになっている行は、左から右に実行します。

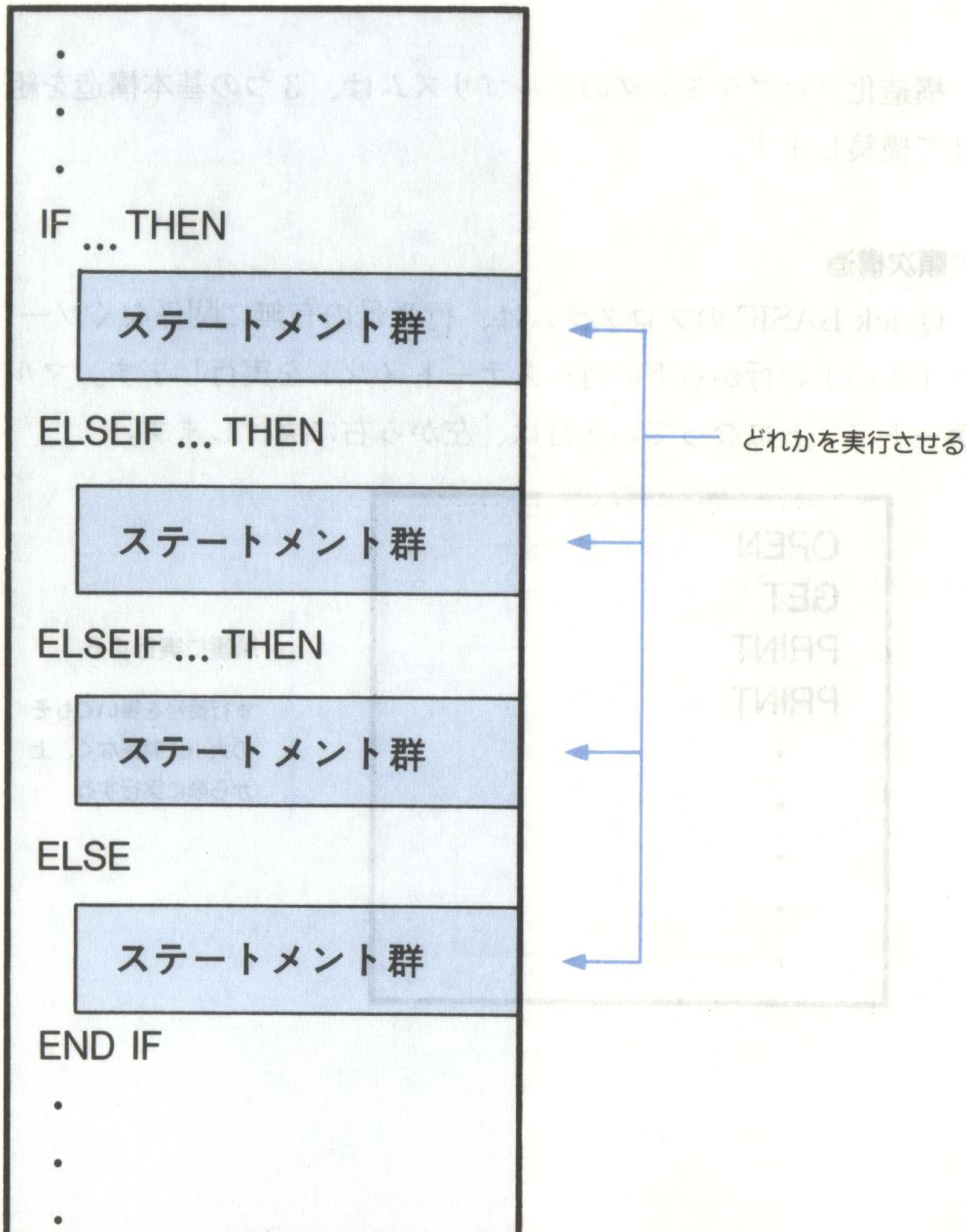


■分岐構造

ある条件を満たす場合だけ、択一的に実行させるためには、つぎのステートメントを使います。

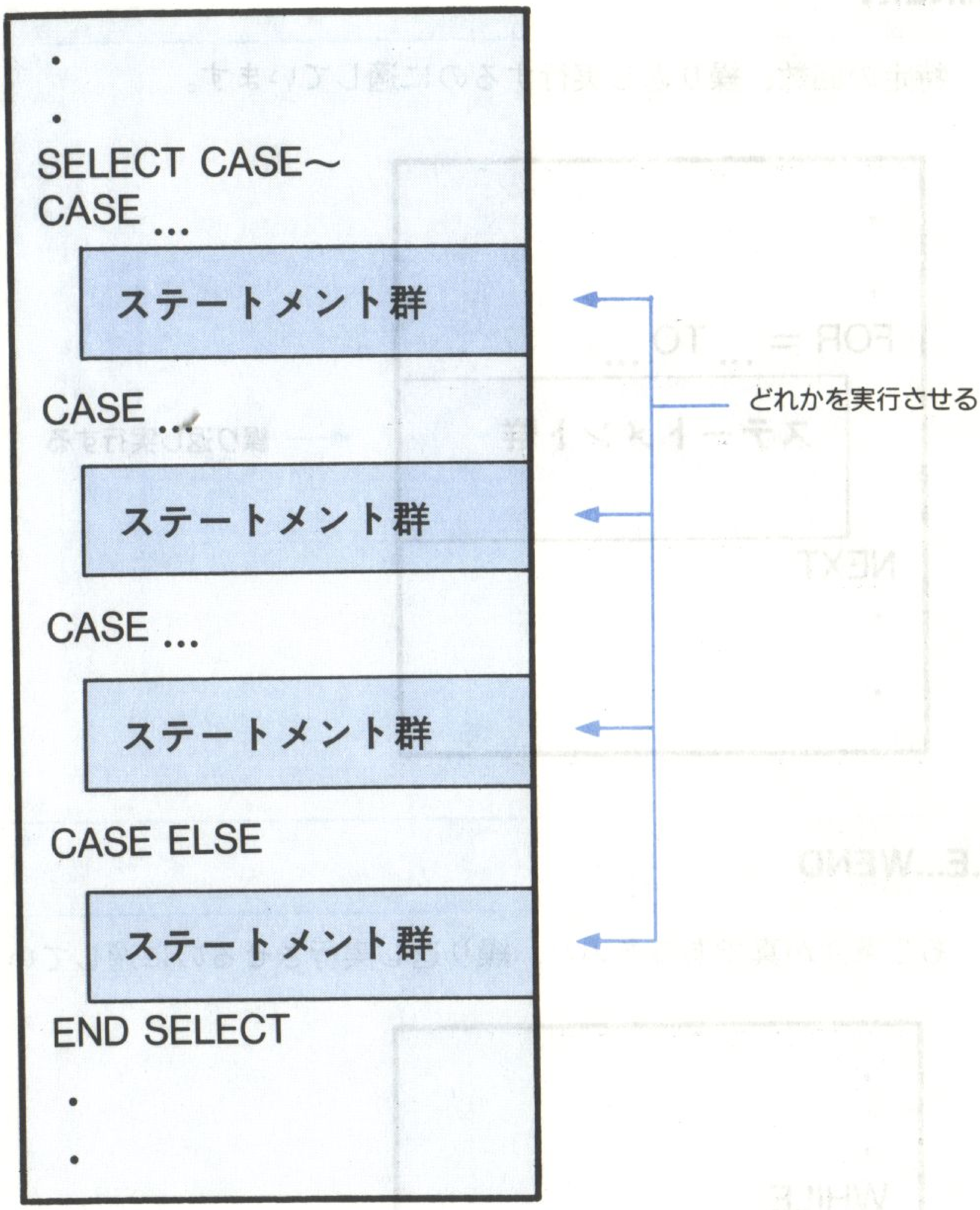
## IF...THEN...ELSE...END IF

分岐の条件を異なったパラメーターを使って判断したいときに適しています。



## SELECT CASE...END SELECT

分岐の条件を同一のパラメーターが異なった値になるときに判断するのに適しています。

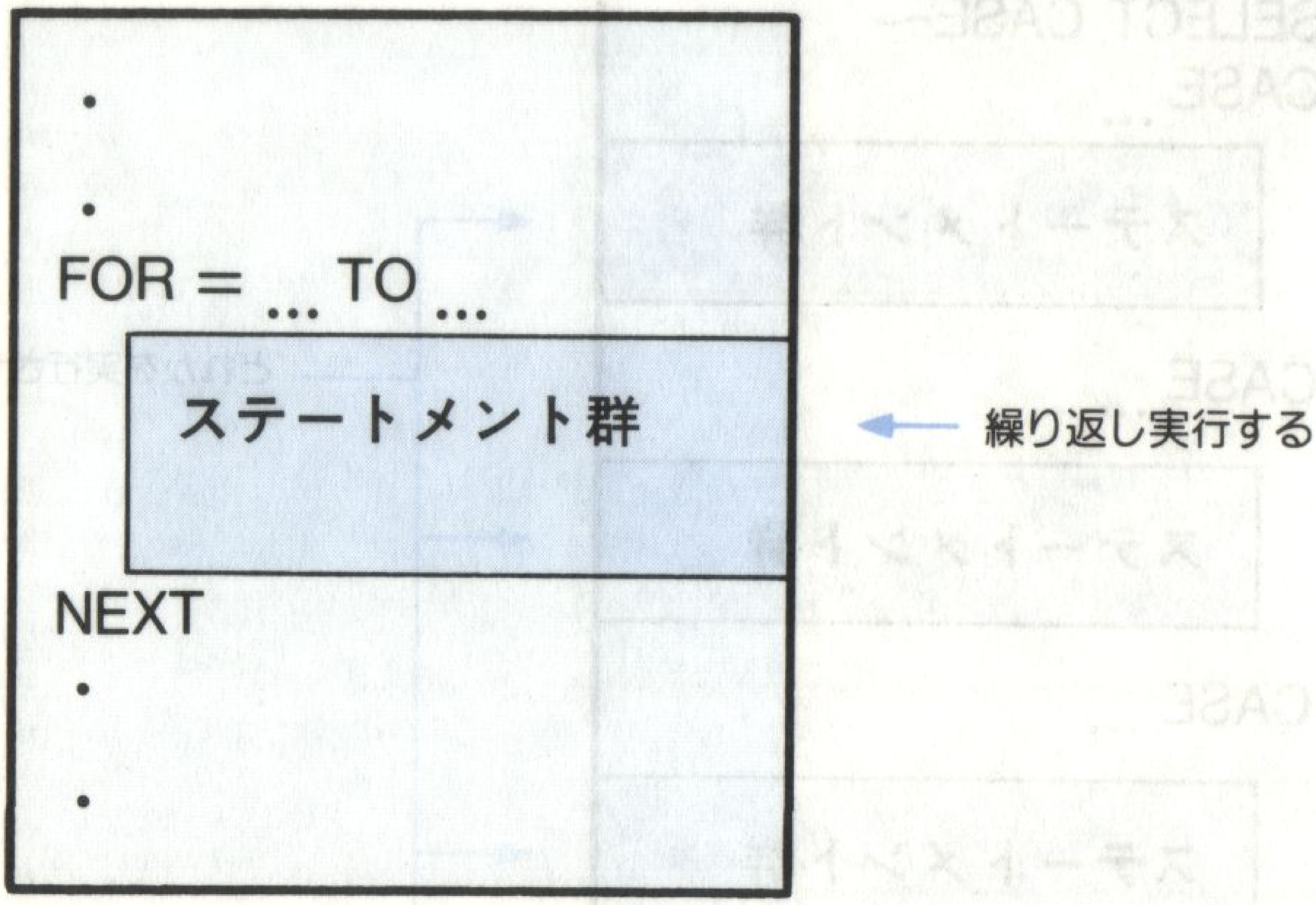


## ■反復構造

ある条件を満たしているあいだ、繰り返し同じ部分を実行させるためには、つぎのステートメントを使います。

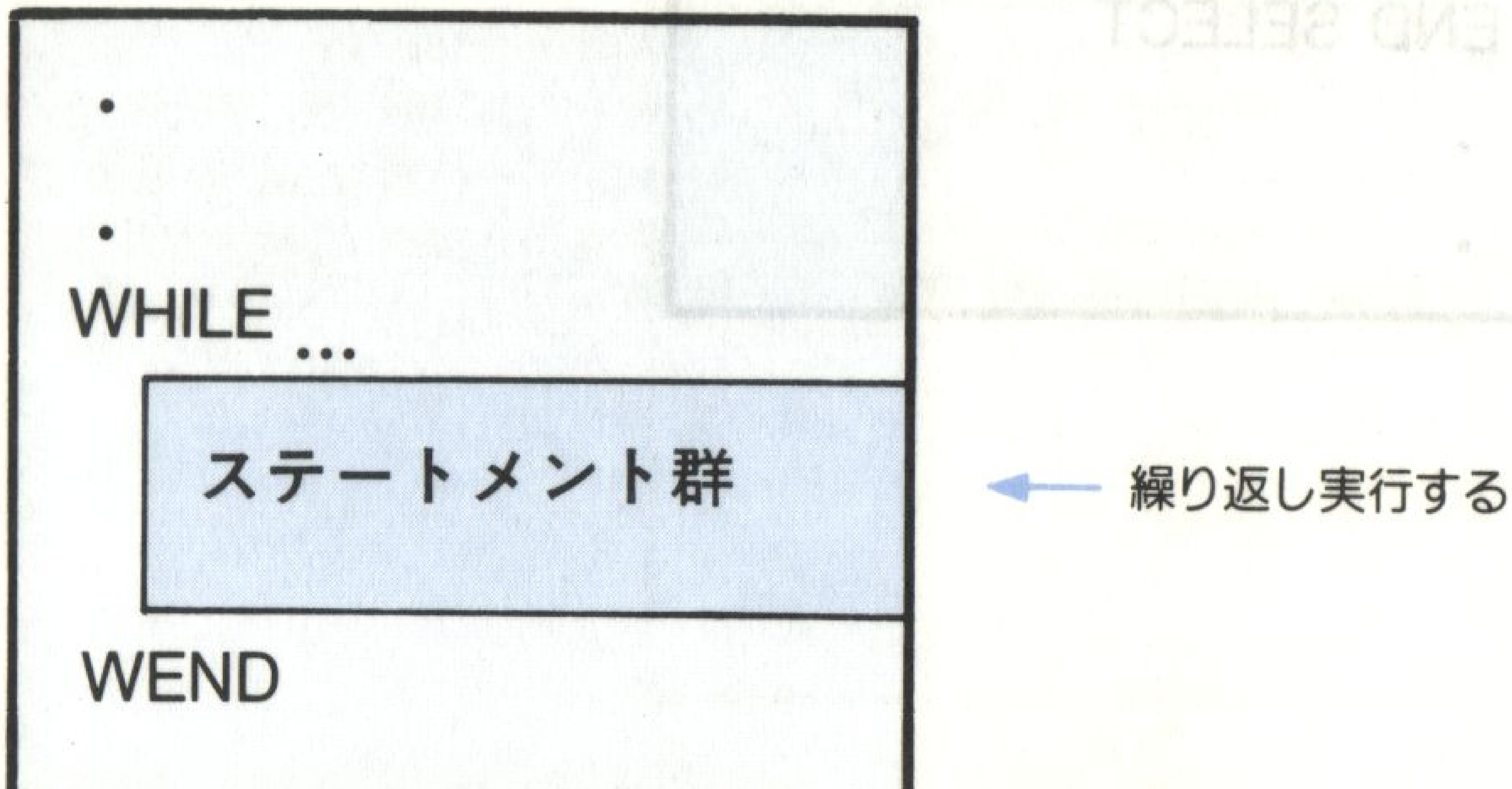
### FOR...NEXT

特定の回数、繰り返し実行するのに適しています。



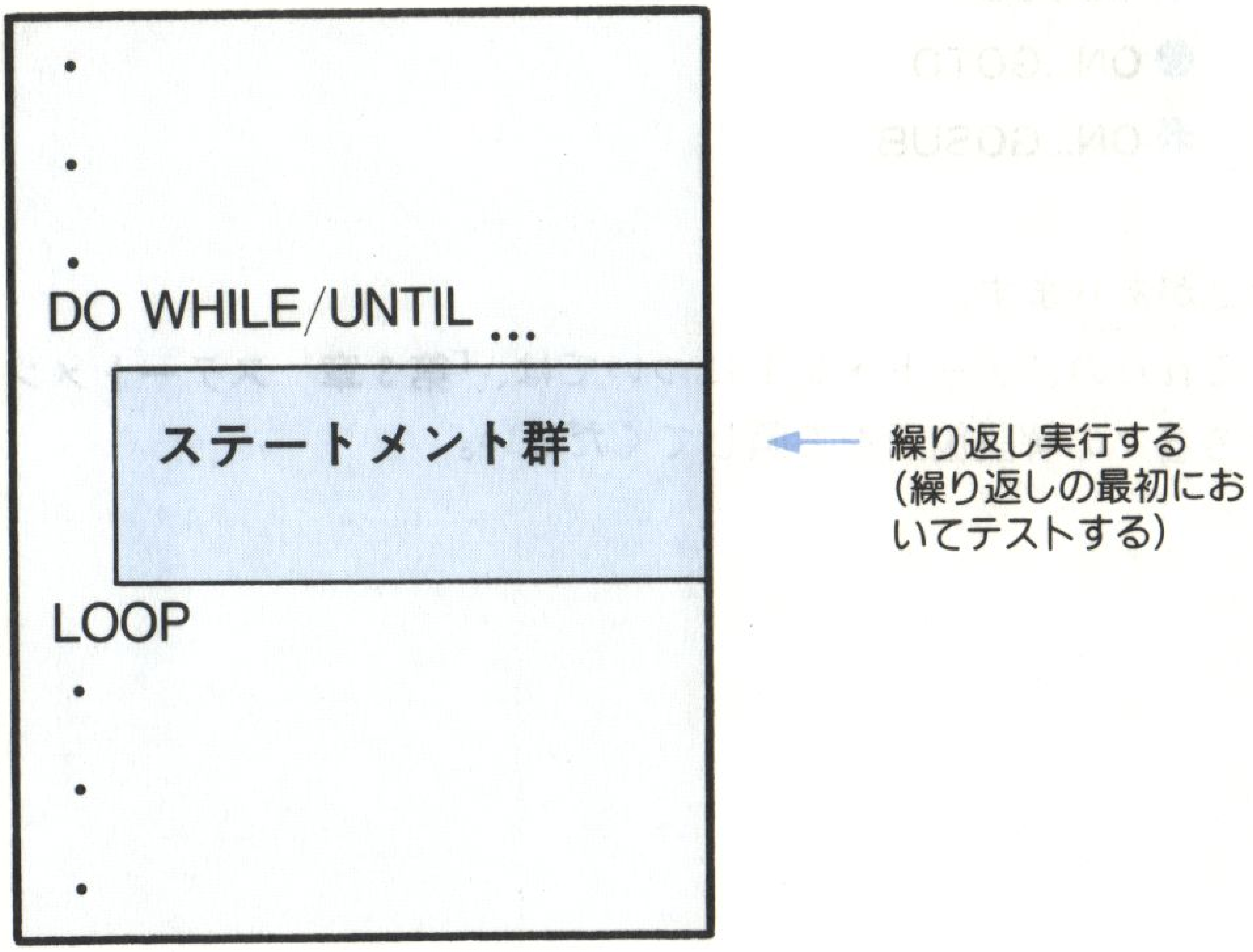

### WHILE...WEND

ある条件が真であるあいだ、繰り返し実行させるのに適しています。

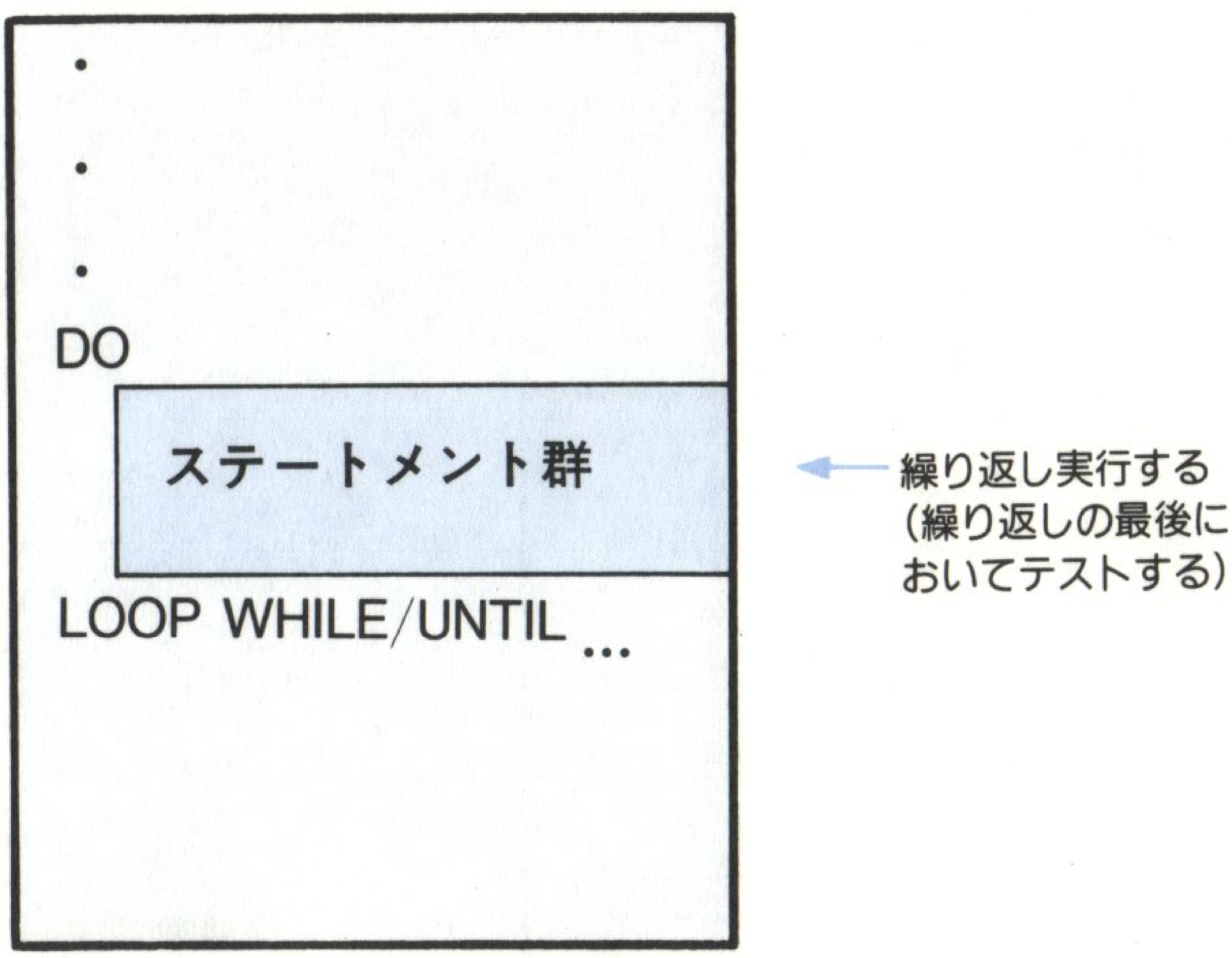


## DO...LOOP

ある条件が真であるあいだ、繰り返し実行させるのに適しています。




または



他にも、分岐するためのステートメントとして、

- GOTO
- GOSUB
- ON...GOTO
- ON...GOSUB

などがあります。

これらのステートメントについては、**「第3章　ステートメント」**における各々の解説箇所を参照してください。

フローの構造化

# 2・4

# プログラムの構造化



## プログラムとモジュール

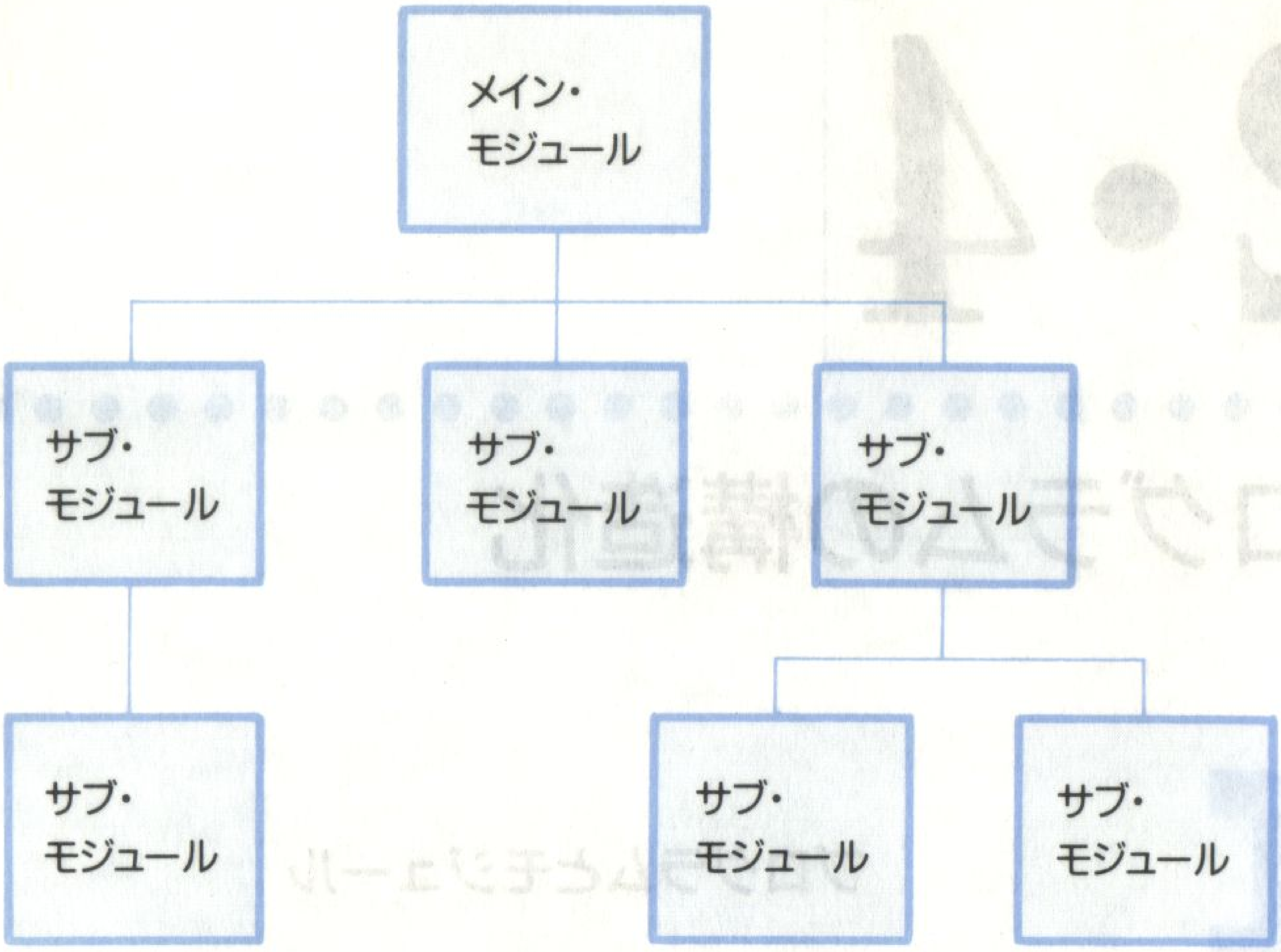
ある問題を解決するために開発したものを総称して「プログラム」と呼びます。

### ■モジュール

1つ以上のステートメント文が書いてあるものを「モジュール」といいます。

プログラムは、必ず1つ以上のモジュールから成りますが、1つだけのモジュールからなるプログラムを「シングル・モジュール」のプログラムといい、2つ以上のモジュールからなるプログラムを「マルチプル・モジュール」のプログラムといいます。

マルチプル・モジュールの中のモジュールのうち、最初に実行されるものを「メイン・モジュール」といい、これ以外を「サブ・モジュール」といいます。






### ■モジュール・レベル・コードとプロシージャ・レベル・コード

各モジュールは、複数のSUBプロシージャとFUNCTIONプロシージャをもつことができます。

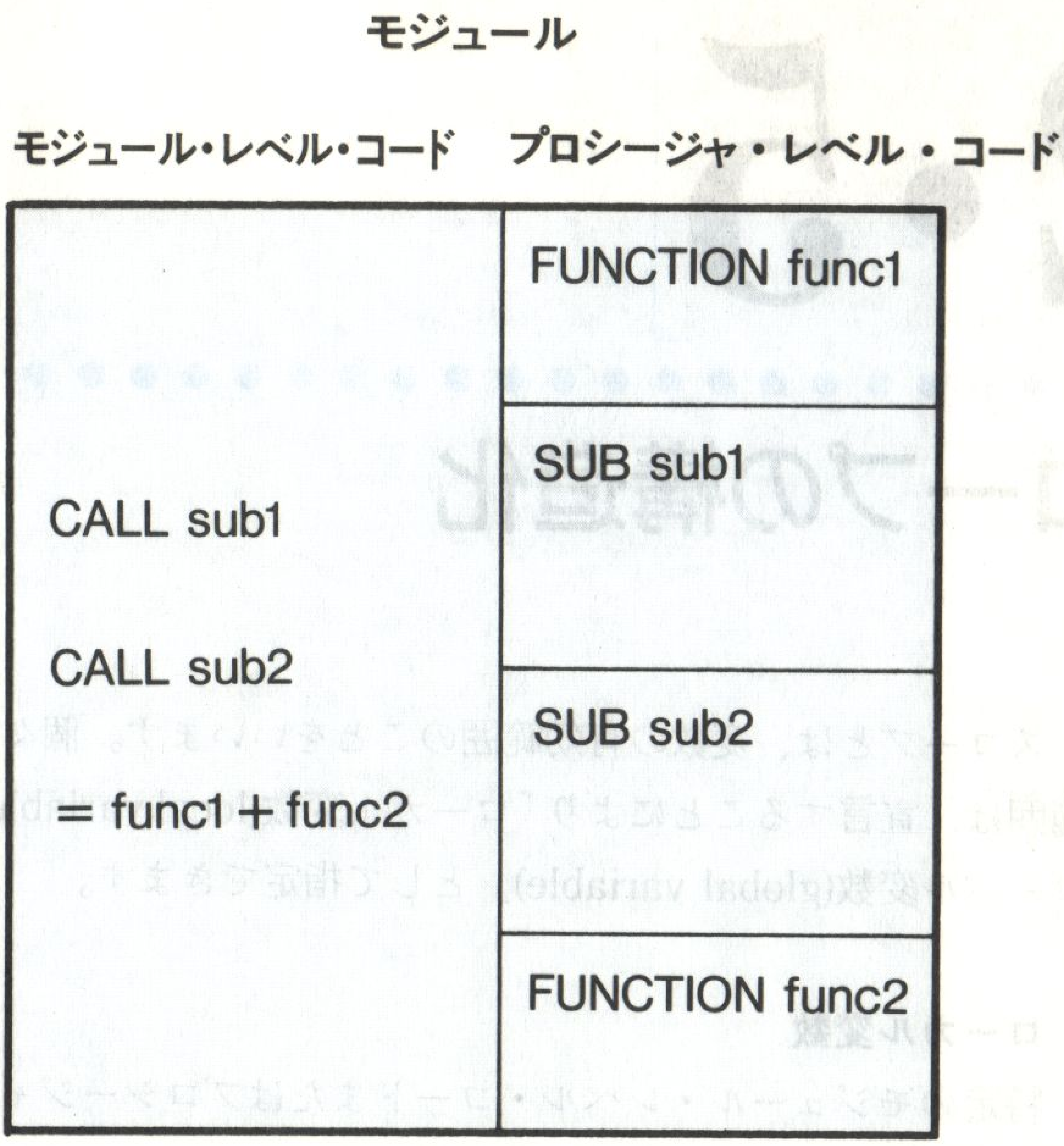
#### ▶ SUBプロシージャ

プログラム中でプログラマー自身が作成したステートメントとして使える。

#### ▶ FUNCTIONプロシージャ

プログラム中でプログラマー自身が作成した関数として使える。

モジュールにおけるプロシージャ以外の基本的な部分を「モジュール・レベル・コード」といいます。また、プロシージャの部分を「プロシージャ・レベル・コード」といいます。


モジュール
モジュール・レベル・コード
プロシージャ・レベル・コード
CALL sub1
CALL sub2
= func1 + func2
FUNCTION func1
SUB sub1
SUB sub2
FUNCTION func2

# 2・5

# スコープの構造化

スコープとは、変数の有効範囲のことをいいます。個々の変数の有効範囲は、宣言することにより「ローカル変数(local variable)」または「グローバル変数(global variable)」として指定できます。

▶ **ローカル変数**

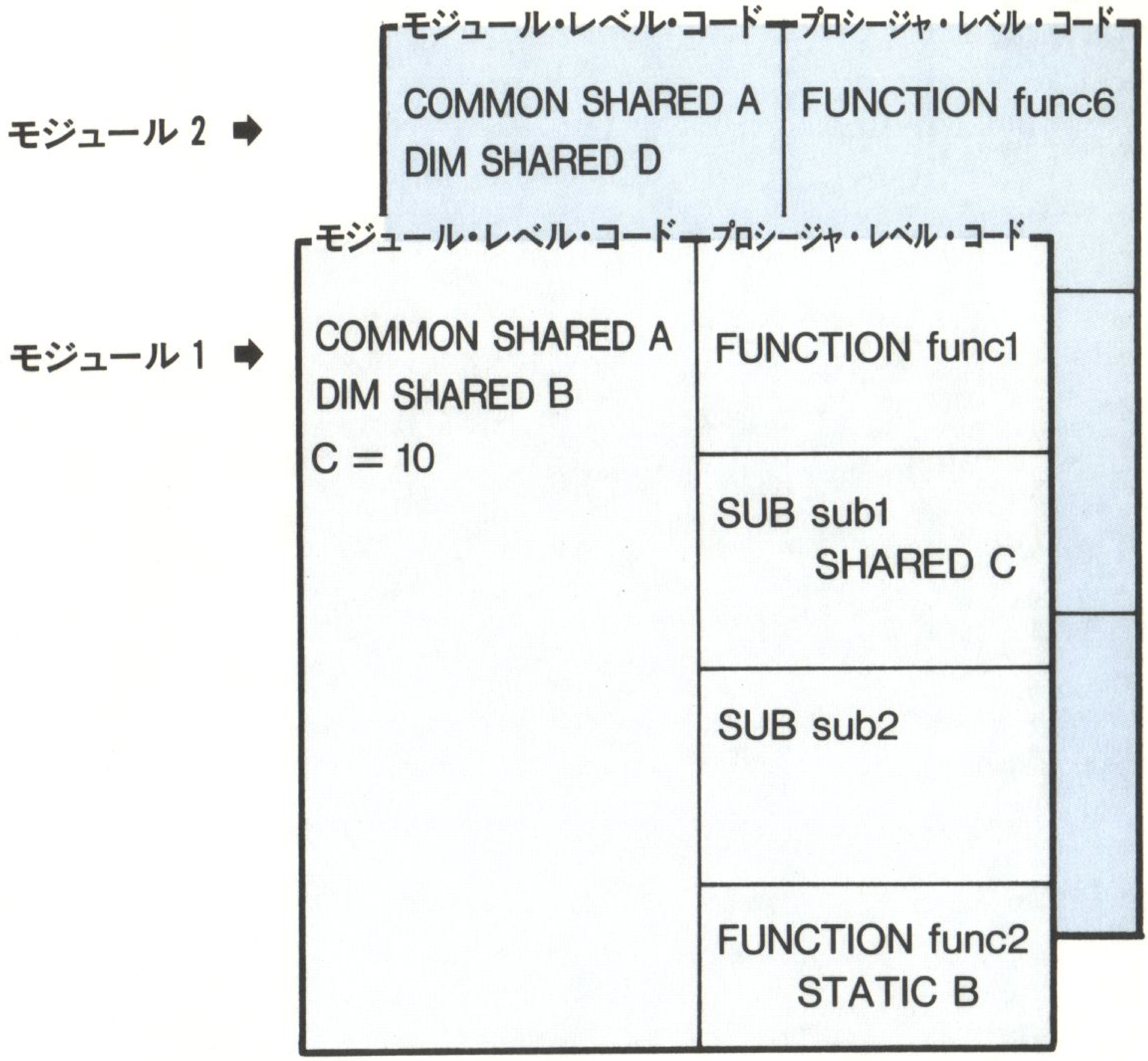
特定のモジュール・レベル・コードまたはプロシージャ内だけで有効な変数としてローカル変数を宣言できる。

▶ **グローバル変数**

モジュール・レベル・コードまたはプロシージャ内に共通して有効な変数としてグローバル変数を宣言できる。

特に宣言しないその変数は、それを最初に使用したモジュール・レベル・コード内、あるいはプロシージャ内でのみ有効になります。しかしながら、複数のブロックに共通して有効であるようにしたり、あるいはグローバル変数と同じ名前のローカル変数を使えるようにするには、つぎのステートメントを使い分けます。

| 目　　的 | ステートメント | 例の変数 |
|---|---|---|
| 複数のモジュール・レベル・コード間で共通にする（そこに含まれるプロシージャにも共通） | COMMON SHARED | A |
| モジュール・レベル・コードとそのすべてのプロシージャ間で共通にする | DIM SHARED<br>REDIM SHARED | A、B<br>D |
| モジュール・レベル・コードと特定のプロシージャ間だけで共通にする | SHARED | C |
| グローバル変数と同名のローカル変数をプロシージャ内で使う | STATIC | func2のB |



# [第3章] ステートメント

この章では、QuickBASICが標準として提供している
ステートメントのすべてをアルファベット順に説明します。
ステートメントについて機能別に知りたいときは、
付録の「機能別ステートメント・関数索引」を利用してください。

3

# この章の読み方

この章では、Quick BASIC(Ver.4.2)における全ステートメントをアルファベット順に要約します。各ステートメントは、以下のスタイルで記述します。

■**ステートメント名**

機能についての要約が記述されています。

■**機能**

働きや用途について記述されています。

■**書式**

完全なシンタックス(構文)が記述されています。

---

例1：**FOR　カウンターj　=　初期値　TO　終了値　［STEP　増分］**

⋮

**NEXT　［カウンターi［, カウンターj...］］**

---

例2：**GOSUB　{行番号 | 行ラベル}**

---

▶書式中の記号の記述と意味

| 用　　途 | 文字・記号 | 例 |
|---|---|---|
| そのまま書くべき箇所 | 大英文字および<br>,　”　-　;　／ | FOR, =, TO<br>STEP, NEXT |
| 適当なものに置き換えて書くべき箇所 | 日本語の部分 | カウンターj<br>初期値, 終了値, 増分 |
| 省略可能な箇所 | [ ] | [STEP　増分] |
| どれか1つを選ぶべき箇所 | { \| } | {行番号 \| 行ラベル} |
| 反復的に使える箇所 | ... | カウンターj ... |
| 省略されている箇所 | ⋮ | ⋮ |

■引数とオプション

引数およびオプションの機能や意味について記述されています。

■文例

実際の文例を掲げてあります。

■ノート

パラメーターやオプションについての補足的な説明や使用するときの注意点について記述されています。

# BEEP

## ビープ音

■機能

ビープ音を出す

■書式

**BEEP**

■引数とオプション

なし

■文例

```
FOR i = 1 TO 3          ' ピッ，ピッ，ピッ，と3回鳴らす
  BEEP
  FOR j = 1 TO 50       ' ダミー・ループ
  NEXT j
NEXT i
```

■ノート

▶ BEEP ステートメントは、

PRINT　CHR$(7)

と同じ機能を持っている

# BLOAD

## メモリー・イメージ・ファイルからのロード



■機能

メモリー・イメージ・ファイルの内容を指定したメモリーにロードする

■書式

**BLOAD　ファイル名　[，オフセット値]**

■引数とオプション

| 引数とオプション | 機　能 |
|---|---|
| ファイル名 | 読み込みたいメモリー・イメージ・ファイルの名前を文字式で指定する。<br>入力デバイスとしての"COMn："も使える。 |
| オフセット値 | 読み込みたいメモリーの先頭アドレスのオフセットを数値式で指定する。省略すると、保存したときのアドレスになる。 |

■文例

```
BLOAD "CatImage.Mem"
```

■ノート

▶書き出すには、BSAVE ステートメントを使う。

# BSAVE

## メモリー・イメージ・ファイルへのセーブ

### ■機能

メモリー内容をディスク上にメモリー・イメージ・ファイルとしてセーブする

### ■書式

**BSAVE　ファイル名，オフセット値，サイズ**

### ■引数

| 引数 | 機　　能 |
|---|---|
| ファイル名 | 書き出したいメモリー・イメージ・ファイルの名前を文字式で指定する。 |
| オフセット値 | 書き出したいメモリーの先頭アドレスのオフセットを数値式で指定する。 |
| サイズ | 書き出したいメモリーの大きさのバイト数を数値式で指定する。 |

### ■文例

```
BSAVE "CatImage.Mem"
```

### ■ノート

▶読み込むには、BLOADステートメントを使う。

▶サイズは、0〜65535の範囲で指定しなければならない。

# CALL [QuickBASIC]

## SUBプロシージャの呼び出し

■機能

任意のSUBプロシージャを呼び出す(コールする)

■書式

**CALL　プロシージャ名　[(引数リスト)]**

■引数とオプション

| 引数とオプション | 機　　能 |
|---|---|
| プロシージャ名 | 呼び出したいSUBプロシージャの名前を文字式で指定する。 |
| 引数リスト | そのSUBプロシージャに渡したい引数のリストを"()"の中に指定する。 |

■文例①

```
productCode$ = "QB42"
isStatus% = 0
CALL FindOneRecord(productCode$, isStatus%)
```

■文例②

```
DECLARE SUB FindOneRecord (aCode$, aResponse%)
    .
    .
    .
productCode$ = "QB42"
isStatus% = 0
FindOneRecord productCode$, isStatus%
```

CALL [Quick BASIC]

■文例③

```
fileNumber% = 3
productCode$ = "QB42"
isStatus% = 0                ( )で囲んで式として渡す
CALL FindOneRecord((fileNumber%), productCode$, isStatus%)
```

■文例④

```
DIM salesOfMonth (1 TO 12)
    .
    .
    .
productCode$ = "QB42"
isStatus% = 0
CALL FindOneRecord(productCode$, salesOfMonth(), isStatus%)
```

■ノート

▶"CALL"の部分は、DECLAREステートメントを使ってそのSUBプロシージャを宣言すれば省略できる(文例②)。

▶引数は、参照渡し(call by reference)されるので、プロシージャ内での書換えが可能である。

▶引数を、値による渡し(call by value)にしたいときは、式として渡すことにより可能となるので、呼び出し時の値は変更されない(文例③)。

▶引数リストの中で配列変数を渡すときは、その変数名に続けて()のみを付ける(文例④)。

▶参照ステートメント：DECLARE、SUB

# CALL/CALLS [他の言語]

## 他の言語で書いたプロシージャの呼び出し

■機能

他の言語で書いたプロシージャを呼び出す

■書式

① CALL　プロシージャ名　[(引数リスト)]

② CALLS　プロシージャ名　[(引数リスト)]

■引数とオプション

| 引数とオプション | 機　能 |
|---|---|
| プロシージャ名 | 呼び出したいプロシージャの名前を指定する。 |
| 引数リスト | 引数があるときは、()で囲んで指定する。<br>1つの引数は、つぎのシンタックスに従う。<br>[{BYVAL \| SEG}] 引数名<br>BYVAL ... 指定すると値による渡し(pass by value)になるので、引数の値は変更されない。<br>SEG ... 引数のセグメント付きfarアドレスを渡したいときに指定する。<br>BYVALおよびSEGを省略するとnearアドレスの参照渡し(pass by reference)になる。 |

■文例

```
CALL BinarySearch(codeArray(), theCode)
```

## ■ノート

▶"CALL"の部分は、DECLAREステートメントを使ってそのSUBプロシージャを宣言すれば省略できる。

▶参照ステートメント：DECLARE

# CALL ABSOLUTE

## 機械語で書いたプロシージャの呼び出し



### ■機能

機械語で書いたプロシージャをコールする

### ■書式

**CALL ABSOLUTE ([引数リスト,] オフセット)**

### ■引数とオプション

| 引数とオプション | 機　能 |
|---|---|
| 引数リスト | プロシージャに渡す引数を指定する。 |
| オフセット | コード・セグメントにおけるプロシージャのオフセットを指定する。 |

### ■文例

```
CALL ABSOLUTE(VARPTR(DosCTRL(1)))
```

### ■ノート

▶インクルード・ファイル"QB.BI"で宣言されているので、これをインクルードしなければならない。

▶クイック・ライブラリー"QB.QLB"に含まれているので、Quick BASIC を起動するときに、/L オプションを使ってライブラリーを指定しておかなければならない。

# CALL INT86old/CALL INT86xold

## MS-DOSのシステム・コール

### ■機能

MS-DOSのシステム・コールを実行する

### ■書式

① CALL INT86old(INT番号, 配列i, 配列o)

② CALL INT86xold(INT番号, 配列i, 配列o)

### ■引数

| 引数 | 機能 |
|---|---|
| INT番号 | DOSへの割り込み番号を指定する。 |
| 配列i | 割り込み時にレジスタに設定したい値を整数型配列変数に書く。 |
| 配列o | 割り込み実行後のレジスタの値がこの整数型配列変数に保持される。 |

### ■文例

```
'$INCLUDE: 'BIN¥QB.BI'
DECLARE FUNCTION GetDrive$ ()

PRINT "Current Drive = " + GetDrive$ + ":"
END
```

```
FUNCTION GetDrive$
  DIM iRegs(7) AS INTEGER, oRegs(7) AS INTEGER
  ax = 0
  iRegs(ax) = &H1900
  INT86old &H21, iRegs(), oRegs()
  GetDrive$ = CHR$((oRegs(ax) AND &HFF) + 65)
END FUNCTION
```



■ノート

▶インクルード・ファイル"QB.BI"で宣言されているので、このファイルをインクルードしなければならない。

▶ INT86oldおよびINT86xoldは、クイック・ライブラリー"QB.QLB"に含まれているので、Quick BASICを起動するときに、/Lオプションを使ってライブラリーを指定しておかなければならない。

▶セグメント・レジスタDS、ESを使うシステム・コールを行なうには、INT86xoldを使う。

▶ QB.BIにおいてDECLAREステートメントを使って宣言されているので、"CALL"を省略できる。このときの引数は（　）で囲まない。

# CALL INTERRUPT/CALL INTERRUPTX

## MS-DOSのシステム・コール



### ■機能

MS-DOSのシステム・コールを実行する

### ■書式

**① CALL INTERRUPT (INT番号, レジスタi, レジスタo)**

**② CALL INTERRUPTX (INT番号, レジスタi, レジスタo)**

### ■引数

| 引数 | 機　能 |
|---|---|
| INT 番号 | DOS への割り込み番号を指定する。 |
| レジスタi | 割り込み時にレジスタに設定したい値をユーザー定義型変数(RegType または RegTypeX)に書く。 |
| レジスタo | 割り込み実行後のレジスタの値がユーザー定義型変数(RegType または RegTypeX)に保持される。 |

### ■文例

```
'$INCLUDE: 'BIN¥QB.BI'
DECLARE FUNCTION GetDrive$ ()

PRINT "Current Drive = " + GetDrive$ + ":"
END
```

```
FUNCTION GetDrive$
  DIM iRegs AS RegType, oRegs AS RegType
  iRegs.ax = &H1900
  INTERRUPT &H21, iRegs, oRegs
  GetDrive$ = CHR$((oRegs.ax AND &HFF) + 65)
END FUNCTION
```



■ノート

▶ RegTypeおよびRegTypeXは、インクルード・ファイル"QB.BI"で宣言されているので、このファイルをインクルードしなければならない。

▶ INTERRUPTおよびINTERRUPTXは、クイック・ライブラリー"QB.QLB"に含まれているので、Quick BASICを起動するときに、/Lオプションを使ってライブラリーを指定しておかなければならない。

▶ セグメントレジスタDS、ESを使うシステム・コールを行なうには、INTERRUPTXを使う。

▶ QB.BIにおいてDECLAREステートメントを使って宣言されているので、"CALL"を省略できる。このときの引数は（　）で囲まない。

```
TYPE RegType
 ax     AS INTEGER
 bx     AS INTEGER
 cx     AS INTEGER
 dx     AS INTEGER
 bp     AS INTEGER
 si     AS INTEGER
 di     AS INTEGER
 flags  AS INTEGER
END TYPE

TYPE RegTypeX
 ax     AS INTEGER
 bx     AS INTEGER
 cx     AS INTEGER
 dx     AS INTEGER
 bp     AS INTEGER
 si     AS INTEGER
 di     AS INTEGER
 flags  AS INTEGER
 ds     AS INTEGER
 es     AS INTEGER
END TYPE
```

# CHAIN

## プログラムの実行

■機能

ディスク上のプログラムを実行する

■書式

**CHAIN　プログラム名**

■引数

| 引数 | 機　能 |
|---|---|
| **プログラム名** | **実行したいプログラム名を指定する。拡張子を省略して、Quick BASICの環境内でこのステートメントを実行すると".BAS"とみなし、また実行可能形式のプログラムの中では、".EXE"とみなす。** |

■文例

```
CHAIN "WordPlus"
```

■ノート

▶このステートメントは、オープン中のデータ・ファイルをクローズしたり、メモリー上のデータをクリアしたりしない。

▶参照ステートメント：RUN、COMMON

# CHDIR

## ディレクトリーの変更

### ■機能

ディレクトリーを変更する。MS-DOSのCHDIRコマンドと同じ機能を持っている。

### ■書式

**CHDIR　パス名**

### ■引数

| 引数 | 機　能 |
|---|---|
| パス名 | 変更したいパス名を文字式で指定する。 |

### ■文例

```
CHDIR "MAILING"
thePath$ = "B:¥MODULES¥DOSCALL"
CHDIR thePath$
```

### ■ノート

▶ディレクトリの作成は、MKDIRステートメントを使う。

▶ディレクトリの削除は、RMDIRステートメントを使う。

# CIRCLE

## 円、楕円を描く

■機能

円または楕円を描く

■書式

CIRCLE [STEP] (x, y), 半径 [, [カラー]
[, [開始点] [, [終了点] [, 縦横比]]]]

■引数とオプション

| 引数とオプション | 機能 |
|---|---|
| STEP | 現在のグラフィック・カーソルの位置から相対的に x, y の座標を決めたいときに指定する。 |
| x, y | 描く円または楕円の中心の座標を指定する。 |
| 半径 | 中心からの円の半径または楕円の長軸を指定する。 |
| カラー | 円または楕円のカラーをコードで指定する。 |
| 開始点<br>終了点 | 円または楕円の一部だけ描きたいときの最初と最後の角度をラジアンで指定する。 |
| 縦横比 | 楕円を描くときの中心からの縦軸と横軸の比率を指定する(1を超えると縦長になる)。 |

■文例

```
FOR aspect = 1 TO 5 STEP .2
  CIRCLE (200, 200), 100, , , , aspect
NEXT aspect
   .
   .
```

■ノート

▶"STEP"を省略すると絶対座標になる。

▶"開始点"と"終了点"は、-2〜2$\pi$の範囲内でなければならない。

▶"開始点"を省略すると0を設定し、"終了点"を省略すると2$\pi$を設定する。

▶"縦横比"を省略すると1が設定される。

▶描いたあとのグラフィック・カーソルの位置は、中心に移動する。

▶"STEP"を省略すると絶対座標になる。

# CLEAR

## 変数の再初期化/ファイルのクローズ/スタック・サイズの設定

■機能

すべての変数を再初期化し、ファイルをクローズし、スタック・サイズを設定する

■書式

**CLEAR [,, スタック]**

■オプション

| オプション | 機能 |
|---|---|
| スタック | 必要なスタック・サイズをバイト数で指定する。 |

■文例

```
CLEAR , , 2024
```

■ノート

▶プロシージャ・コールやサブルーチン・コールのレベルが深かったり、再帰的なコールを行なうプログラムでは相応のスタック・サイズが必要になる。

▶参照関数：SETMEM、FRE

# CLOSE

## ファイルのクローズ

■**機能**

ファイルまたはデバイスへの入出力をクローズする

■**書式**

**CLOSE [[#] ファイル番号 [, [#] ファイル番号]...]**

■**引数とオプション**

| 引数とオプション | 機能 |
|---|---|
| ファイル番号 | クローズしたいファイルの番号を指定する。 |

■**文例**

```
OPEN "Foods.Dat" FOR OUTPUT AS 1
DO
  INPUT ; "Product = ", aName$
  IF aName$ = "" THEN
    EXIT DO
  END IF
  INPUT " , Sales = ", sales
  WRITE #1, aName$, sales
LOOP
CLOSE 1
```

■**ノート**

▶ファイル番号を省略すると、オープンされているすべてをクローズする。

▶参照ステートメント：OPEN、INPUT #、WRITE #、GET #、PUT #

# CLS

## スクリーンのクリア

■機能

スクリーンをクリアする

■書式

**CLS [{0 | 1 | 2}]**

■オプション

| オプション | 機能 |
|---|---|
| 0 | 表示されているすべてのテキストおよびグラフィックをクリアする。 |
| 1 | VIEW ステートメントにより物理ビューポートが設定されているときは、その範囲内のグラフィックをクリアする。それ以外は、すべてのグラフィックをクリアする。 |
| 2 | テキストだけをクリアする。 |

■文例

```
CLS
```

■ノート

▶オプションを指定しないと、テキストとグラフィックをクリアする。

▶クリアした後、カーソルは、1,1(ホーム・ポジション)の位置に移動する。

# COLOR

## スクリーン表示カラーの設定



■機能

スクリーン表示のカラーを設定する

■書式

**COLOR [前景カラー] [,背景カラー]**

■引数とオプション

| オプション | 機　　能 |
|---|---|
| 前景カラー | 前景のカラーを指定する。 |
| 背景カラー | 背景のカラーを指定する。 |

■文例

```
COLOR 10, 0
LOCATE 1, 25
PRINT ">>>> QuickBASIC <<<<"
```

■ノート

▶テキストのカラーの設定は、以下の意味をもっている。

| コード | カラー |
| --- | --- |
| 0～7 | 黒、青、緑、空色、赤、マジェンタ、黄、白 |
| 8～15 | 反転させる |
| 16～23 | ノーマル表示でブリンクさせる |
| 24～31 | 反転表示でブリンクさせる |

▶グラフィックのカラーの設定は、属性番号としての意味をもっている。

▶カラーの種類は、SCREENステートメントで設定するパレット番号に依存する。

▶カラーのコードと属性番号の対応関係は、PALLETTEステートメントで設定する。

▶参照ステートメント：SCREEN、PALETTE

# COM(n)

## 通信ポートのイベント・トラッピングの設定

■機能

通信ポートのイベント・トラッピングの実行の有無を設定する

■書式

**COM(n)　{ON | OFF | STOP}**

■引数とオプション

| 引数とオプション | 機　　能 |
|---|---|
| n | 通信用のポート番号(1または2)を指定する。 |
| ON | 通信ポートのイベント・トラッピングを有効にする。有効であるあいだに通信データを受信すると、<br>ON COM ステートメント<br>で指定したイベント・トラッピングのサブルーチンを実行する。 |
| OFF<br>STOP | 通信ポートのイベント・トラッピングを無効にする。OFF は、そのイベントを保持しないが、STOP は、保持しているのでトラッピングできるようになると ON COM の処理ルーチンを実行する。 |

■文例

```
COM(1) ON
```

■ノート

▶参照ステートメント：ON　event

COM(n)

# COMMON

## グローバル変数の定義

### ■機能

モジュール間およびプログラム間で共通して使うグローバル変数を定義する

### ■書式

**COMMON [SHARED] [/ブロック名/] 変数 [()] [AS タイプ]**
**[, 変数 [()] [AS タイプ]]...**

### ■引数とオプション

| 引数とオプション | 機能 |
|---|---|
| SHARED | モジュール・レベルの変数をそのモジュールに含まれるすべてのプロシージャとの間で共有化したいときに指定する。 |
| /ブロック名/ | 一部の変数だけを共有したいときにグループ名として付ける。 |
| 変数 | グローバル変数として宣言したい変数を指定する。 |

| | | |
|---|---|---|
| タイプ | 変数のタイプを指定する。このタイプ指定を行なうと変数名には、型宣言文字(%、&、!、#、$)を付ける必要はない。 | |
| | **タイプ名** | **機　能** |
| | INTEGER | 整数型 |
| | LONG | 長整数型 |
| | SINGLE | 単精度型 |
| | DOUBLE | 倍精度型 |
| | STRING | 固定長文字列(n は文字長)<br>STRING　*　n |
| | ユーザー定義タイプ | ユーザー定義タイプ |

## ■文例

```
COMMON SystemLastMenu, SystemLastRec
COMMON SystemWindowPtr, SystemWindowHdr
```

## ■ノート

▶このステートメントは、共有したいモジュールそれぞれにおいて宣言しなければならない。

▶共有するモジュール間において、変数名は同じである必要はないが、宣言するときの順番および型は同じでなければならない。

▶参照ステートメント：CHAIN、DIM、SHARED

# CONST

## シンボリック定数の宣言と定義



■機能

シンボリック定数を宣言および定義する

■書式

**CONST　変数名　=　式, [, 変数名　=　式]...**

■引数

| 引数 | 機　能 |
|---|---|
| 変数名 | 定義したい変数を指定する。 |
| 式 | 定数として代入したい式を指定する。 |

■文例

```
CONST theName$ = "Twenty-One"
CONST aCardMax = 13, typeMax = 4, cardsMax = aCardMax * typeMax
CONST FALSE = 0, TRUE = NOT FALSE
```

■ノート

▶「シンボリック定数」とは、プログラムの実行中に変えたりしない値をもつ変数をいう。

▶変数の型宣言文字を省略すると、その変数の型は式の型になる。

▶式には、リテラル定数およびそれ以前に定義したシンボリック定数、演算式、論理演算式を使えるが、関数や通常の変数は使えない。

▶モジュール内で宣言したシンボリック定数は、そのモジュールに対してローカルになる。

# DATA

## READステートメントで読み込むデータの定義

■機能

READステートメントで読み込まれる定数を定義する

■書式

DATA　定数1　[, 定数2]...

■引数

| 引数 | 機　能 |
|---|---|
| 定数n | READステートメントで読み込まれる定数を定義する |

■文例①

```
DIM dayName(6) AS STRING * 9
FOR n = 0 TO 6
  READ dayName(n)
NEXT n
  .
  .
  .
DATA Sunday, Monday, Tuesday, Wednesday
DATA Thursday, Friday, Saturday
  .
  .
```

■文例②

```
DATA 32, , 45, ""
```

## ■ノート

▶ DATA ステートメントは、モジュール・レベルに書かなければならない。

▶数値型のデータは、READ ステートメントでの対応する変数が数値型ならば数値データとして読み込まれ、また文字列変数ならば文字データとして読み込まれる。

▶文字データの一部として、","、":"、"" 空白などを定義したいときは、その文字列全体を"　"で囲む。

▶ヌル・データを定義するときは、文例②のようにする。

▶参照ステートメント：READ、RESTORE

DATE

# DATE$



## 日付の設定

### ■機能

MS-DOSの日付を設定する

### ■書式

**DATE$ = 日付**

### ■引数

| 引数 | 機　能 |
|---|---|
| 日付 | 設定したい日付を文字式で指定する。<br>時刻は、"YYYY-MM-DD"<br>年(1980～2099)→YYYY<br>月(01～12)→MM<br>日(01～末日)→DD<br>の形式で指定する。 |

### ■文例①

```
DATE$ = "1989-12-31"
```

### ■文例②

```
DATE$ = "89-05-22"
```

### ■ノート

- ▶年の部分は、下2桁でも指定できる(文例②)。
- ▶ MS-DOSのDATEコマンドと同じ機能をもっている。
- ▶ MS-DOS上の現在の日付を知るには、TIME$関数を使う。

# DECLARE [QuickBASIC]

## FUNCTION および SUB の宣言

■機能

FUNCTION および SUB を宣言する。引数があるならば、その型も宣言する。

■書式

**DECLARE {FUNCTION | SUB} プロシージャ名 [([引数リスト])]**

■引数とオプション

| 引数とオプション | 機能 |
|---|---|
| FUNCTION<br>SUB | 宣言したいプロシージャのタイプに従って、どちらかを指定する。 |
| プロシージャ名 | 宣言したいプロシージャの名前を指定する。 |
| 引数リスト | 引数があるときは、()で囲んで指定し、ないときは()のみを書く。 |

■文例

```
DECLARE FUNCTION BibarySearch% (itArray(), keyData)
DECLARE SUB ShowMenu (menuNo AS INTEGER)
```

■ノート

▶ CALL ステートメントを使わずに他のモジュールにおけるプロシージャをコールする(呼び出す)には、使用するモジュールにおいて、このステートメントを使って宣言しなければならない。

▶引数リストにおける変数名は、実際のプロシージャの引数名と同じである必要はないが、その型・数・並びは、同じでなければならない。

▶引数リストにおける型の宣言は、

・型宣言文字　　(%、&、!、#、$)

・AS タイプ

(INTEGER、LONG、SINGLE、DOUBLE、STRING、ユーザー定義型)

のどちらかを使って行なう。

▶参照ステートメント：CALL、FUNCTION、SUB

DECLARE [Quick BASIC]

# DECLARE［他の言語］

## 他の言語で書いたプロシージャの宣言



■機能

他の言語で書いたプロシージャのコーリング・シーケンスを宣言する

■書式

**DECLARE　{FUNCTION | SUB}　プロシージャ名**
**[CDECL] [ALIAS "別名"] [([引数リスト])]**

■引数とオプション

| 引数とオプション | 機　能 |
|---|---|
| FUNCTION<br>SUB | 宣言したいプロシージャのタイプに従って、どちらかを指定する。 |
| プロシージャ名 | 宣言したいプロシージャの名前を指定する。 |
| CDECL | C 言語で作成したプロシージャを使う場合、引数を渡すときの並びを左右入れ替えて渡すことを指定する。 |
| 別名 | プロシージャ名の代わりに別名を使うときに指定する。 |

| | |
|---|---|
| 引数リスト | 引数があるときは、()で囲んで指定し、ないときは()のみを書く。<br>1つの引数は、つぎのシンタックスに従う。<br>[{BYVAL \| SEG}] 引数名 [AS タイプ]<br>BYVAL...指定すると値による渡し(pass by value)になるので、引数の値は変更されない。(INTEGER, LONG, SINGLE, DOUBLEのみ可)<br>SEG ...引数のセグメント付きfarアドレスを渡したいときに指定する。<br>タイプ ...つぎのものが使える。<br>INTEGER<br>LONG<br>SINGLE<br>DOUBLE<br>STRING<br>ユーザー定義型<br>ANY(型を問わない) |

■文例

```
DECLARE FUNCTION GetCmd CDECL ALIAS "get_cmd"(BYVAL menu%,item%)
DECLARE SUB Mouse CDECL (ax%, bx%, cx%, dx%)
```

■ノート

▶ CALLステートメントを使わずに他のモジュールにおけるプロシージャをコールする(呼び出す)には、使用するモジュールにおいて、このステートメントを使って宣言しなければならない。

▶ "別名"は、Quick BASICでは認められない"_"が、C言語の関数名に含まれているときに使う。

▶ 参照ステートメント：CALL、FUNCTION、SUB

# DEF　FNname

## 関数の定義

■機能

関数を定義する。1行で定義するには、書式①を使う。2行以上使って定義するときは、書式②を使う。

■書式

① **DEF　FNname［(引数リスト)］　=　式**

② **DEF　FNname［(引数リスト)］**

⋮

**FNname　=　式**

⋮

**END　DEF**

■引数とオプション

| 引数とオプション | 機　能 |
|---|---|
| FNname | 定義したい関数名を指定する。 |
| 引数リスト | 引数があるときは、それを()で囲んで渡す。 |
| 式 | その関数が返す値を指定する。 |

■文例①

```
DEF fnIncrement% (i%) = i% + 1
n = 1
n = fnIncrement%(n)
```

■文例②

```
DEF fnIncrement% (i%)
    fnIncrement% = i% + 1
END DEF
n = 1
n = fnIncrement%(n)
```

■ノート

▶"FNname"は、FNの部分を含めて40文字以内であり、nameの先頭は文字でなければならない。

▶戻り値の型は、"FNname"にその型宣言文字を付けて行なう。

▶戻り値の型が異なれば、別の関数として取り扱われる。

▶引数として、配列・ユーザー定義型・固定長文字列を渡せない。

▶書式②では、戻り値は、そのFNnameに代入することにより返される。これを行わないと、0またはヌルが返される。

▶書式②において、END DEF以前に強制的に抜け出るには、EXIT DEFステートメントを使う。

▶参照ステートメント：FUNCTION

# DEF SEG

## セグメント・アドレスの設定



### ■機能

セグメント・アドレスを設定する

### ■書式

**DEF SEG [= アドレス]**

### ■オプション

| オプション | 機　　能 |
|---|---|
| アドレス | 設定したいセグメント・アドレス(0～65535)を指定する。 |

### ■文例

```
DEF SEG = &H7700
```

### ■ノート

▶"アドレス"を省略すると、データ・セグメント(DS)が設定されます。

▶参照ステートメント

BLOAD、BSAVE、CALL ABSOLUTE、POKE

▶参照関数：PEEK、VARSEG、VARPTR

# DEFINT/DEFLNG/DEFSNG/DEFDBL/DEFSTR

## 型の設定

### ■機能

変数、DEF　FN、および FUNCTION の型を設定する。設定できる型は、整数型・長整数型・単精度型・倍精度型・文字列型である。

### ■書式

DEFINT　文字範囲［，文字範囲］...
DEFLNG　文字範囲［，文字範囲］...
DEFSNG　文字範囲［，文字範囲］...
DEFDBL　文字範囲［，文字範囲］...
DEFSTR　文字範囲［，文字範囲］...

### ■引数

| 引数 | 機能 |
| --- | --- |
| 文字範囲 | 設定したいアルファベット文字の範囲を<br>　文字-文字<br>の形式で指定する。 |

| ステートメント | 型 |
| --- | --- |
| DEFINT | 整数型 |
| DEFLNG | 長整数型 |
| DEFSNG | 単精度型 |
| DEFDBL | 倍精度型 |
| DEFSTR | 文字列型 |

■文例

```
DEFINT A-C
DEFLNG L, X-Z
DEFSTR O-P, S
```

■ノート

▶アルファベットの大文字と小文字の区別はない。

▶参照ステートメント：DIM

# DIM

## 配列と変数の型の宣言

■機能

配列と変数の型を宣言し、必要なメモリー・スペースを割り当てる。また、モジュール・レベル内で共通に使えるグローバル変数を宣言する。

■書式

**DIM　[SHARED]　変数 [(添字...)] [AS　タイプ]**
**　　[, 変数 [(添字...)] [AS　タイプ]]...**

■引数とオプション

| 引数とオプション | 機　能 |
|---|---|
| SHARED | モジュール・レベルの変数をそのモジュールに含まれるすべてのプロシージャとの間で共有化したいときに指定する(グローバル変数として宣言する)。 |
| 変数 | 宣言したい変数を指定する。 |
| 添字 | 変数を配列として宣言するときにその次元と範囲を指定する。<br>添字は、以下のようにして最大60次元まで指定できる。<br>　(添字 [, 添字, ...])<br>1つの添字の範囲(-32768～32767)は、<br>　　[下限 TO]　上限<br>の形式で指定する。 |

| | |
|---|---|
| タイプ | 変数のタイプを指定する。このタイプ指定を行なうと変数名には、型宣言文字(%、&、!、#、$)を付ける必要はない。 |

| タイプ名 | 機　　能 |
|---|---|
| INTEGER | 整数型 |
| LONG | 長整数型 |
| SINGLE | 単精度型 |
| DOUBLE | 倍精度型 |
| STRING | 固定長文字列(n は文字長)<br>STRING　*　n |
| ユーザー定義タイプ | ユーザー定義タイプ |

## ■文例

```
TYPE DateType
  Year AS INTEGER
  month AS INTEGER
  day AS INTEGER
END TYPE
DIM menuItem(5, 10) AS STRING * 10,  lastDayOfMonth(1 TO 12)
DIM birthday AS DateType
```

## ■ノート

▶参照ステートメント

SHARED、COMMON、STATIC、OPTION BASE

▶参照関数：LBOUND、UBOUND

# DO...LOOP

## 繰り返し実行

■**機能**

DO と LOOP のあいだのステートメントを繰り返し実行する

■**書式**

**① DO　[{WHILE | UNTIL} 論理式]**
**　　[ステートメント・ブロック]**
**LOOP**

**② DO**
**　　[ステートメント・ブロック]**
**LOOP　[{WHILE | UNTIL} 論理式]**

■**オプション**

| オプション | 機　能 |
|---|---|
| WHILE | WHILE は、ステートメント・ブロックをその論理式が真であるあいだ、実行させたいときに指定する。 |
| UNTIL | UNTIL は、ステートメント・ブロックをその論理式が真になるまで、実行させたいときに指定する。 |
| ステートメント・ブロック | 繰り返し実行したいステートメント |

■**文例①：3回繰り返し実行する**

```
cnt = 1
DO WHILE cnt <= 3
  cnt = cnt + 1
LOOP
```

```
cnt = 1
DO UNTIL cnt > 3
  cnt = cnt + 1
LOOP
```

DO...LOOP

■文例②

```
cnt = 1
DO
  cnt = cnt + 1
LOOP WHILE cnt <= 3
```

```
cnt = 1
DO
  cnt = cnt + 1
LOOP UNTIL cnt > 3
```

■ノート

▶書式①では、最初にテストされるので、そのループを初めて実行するときに論理式が偽ならば、一度もそのループを実行しない。

▶書式②では、最後にテストされるので、そのループは少なくとも1回は実行される。

▶"WHILE　論理式"または"UNTIL　論理式"の部分を省略すると無限ループになる。

▶論理式のテスト結果に関係なく、ループから抜け出るには、EXIT　DOステートメントを使う。

▶参照ステートメント：WHILE、FOR

# DRAW

## マクロ・コマンドを使った描画



■機能

マクロ・コマンドで定義したオブジェクトをグラフィック・スクリーンに描く

■書式

**DRAW　マクロ文字式**

■引数

| 引数 | 機　能 |
|---|---|
| マクロ文字式 | グラフィックのオブジェクトを描くためのコマンドとパラメータを文字列として指定する。 |

■文例

```
FOR angle = 0 TO 360 STEP 30
  DRAW "TA=" + VARPTR$(angle) + "BM300,200 R100 D100 L100 U100"
NEXT
```

■ノート

▶マクロ・コマンドとして、以下のものが用意されている。

| コマンド | 機　能 |
|---|---|
| B | ペン先の移動だけを行なう。 |
| N | 元の位置に戻る。 |

| | |
|---|---|
| L [n] | n単位だけ左に移動する。 |
| R [n] | n単位だけ右に移動する。 |
| U [n] | n単位だけ上に移動する。 |
| D [n] | n単位だけ下に移動する。 |
| H [n] | n単位だけ左上に移動する。 |
| G [n] | n単位だけ左下に移動する。 |
| E [n] | n単位だけ右上に移動する。 |
| F [n] | n単位だけ右下に移動する。 |
| Mx, y | x, yに符号を付けないと座標(x, y)に移動する。<br>x, yに符号を付けると現在の座標から相対的に移動する(xとyの間に", "が必要)。 |
| An | 以後描くものを回転させる。nは、0〜3で指定し、それぞれ0°、90°、180°、270°を意味する。 |
| TAn | 以後描くものを回転させる。nは、その角度(−360°〜360°)を指定する。 |
| Cn | カラー属性をnで指定する。 |
| Sn | スケールの倍率をn(1〜255)で指定する。 |
| Pp, b | 塗りつぶすカラーをpで指定し、境界線カラーをbで指定する(pとbの間に", "が必要)。 |
| X | マクロの一部として文字列変数を使うときに指定する。<br>文字列変数の部分にはVARPTR$を使う。 |
| cmd = | マクロの一部として数値変数を使うときに指定する。<br>数値変数の部分にはVARPTR$を使う。 |

※ B、Nは、移動コマンドの前に付けて使う。

※ X, cmd =は、BASICAとの互換性を保つためにもある。

▶参照ステートメント：COLOR、PALETTE、PAINT

# END

## プログラム、プロシージャ、ブロックの終了

### 機能

プログラム、プロシージャ、またはブロックを終了する

### 書式

**END　[{DEF | FUNCTION | IF | SELECT | SUB | TYPE}]**

### オプション

| オプション | 機能 |
|---|---|
| DEF | DEF　FN による関数の定義を終える。 |
| FUNCTION | FUNCTION プロシージャの定義を終える。 |
| IF | IF による分岐を終える。 |
| SELECT | SELECT　CASE による分岐を終える。 |
| SUB | SUB プロシージャの定義を終える。 |
| TYPE | ユーザー定義型の変数の定義を終える。 |

### 文例

```
 ・
 ・
END
```

### ノート

▶オプションを省略したときは、プログラムの実行は終わる。統合環境下で実行したプログラムは、そのプログラムを終了する。また、実行可能ファイル形式(.EXE)のプログラムを実行したときは、そのプログラムを終了しDOSへ制御を戻す。

▶参照ステートメント

DEF　FN、FUNCTION、IF、SELECT、SUB、TYPE、STOP

# ENVIRON

## 環境変数の設定

### ■機能

MS-DOSの環境変数を設定・変更する

### ■書式

**ENVIRON　文字式**

### ■引数

| 引数 | 機　能 |
|---|---|
| 文字式 | 設定もしくは変更した環境変数名とその文字列を1つの文字式として指定する。 |

### ■文例

```
ENVIRON "ERROR=ON"
ENVIRON "PATH=A:¥;B:¥;C:¥;"
```

### ■ノート

▶環境変数については、MS-DOSのSET、PATH、PROMPTコマンドを参照のこと。

▶参照ステートメント：SHELL

▶参照関数：ENVIRON$

# ERASE

## 静的配列の再初期化／動的配列の解放



■機能

静的配列を再初期化または動的配列を解放する

■書式

**ERASE　配列名［，配列名...］**

■引数

| 引数 | 機　能 |
|---|---|
| 配列名 | 再初期化したい静的配列変数名または解放したい動的配列変数名を指定する。 |

■文例

```
'$STATIC
DIM stock(1 TO 100)
'$DYNAMIC
DIM temporary(20, 50)
   .
   .
ERASE stock, temporary
   .
   .
```

■ノート

▶静的配列の再初期化では、その変数の内容が初期化されるのに対し、動的配列の解放では、その変数自体が消去される。したがって、その動的配列変数を使うには、DIM または REDIM ステートメントで宣言しなければならない。

▶参照ステートメント：DIM、REDIM

▶参照コマンド：$DYNAMIC、$STATIC

# ERROR

## エラーのシミュレート



### ■機能

エラーの発生をシミュレートする

### ■書式

**ERROR　エラー・コード**

### ■引数とオプション

| 引数 | 機　能 |
|---|---|
| エラー・コード | シミュレートしたいエラーのコード(1〜255)を指定する。 |

### ■文例

```
ON ERROR GOTO ErrorHandleRoutine
    .
    .
ErrorHandleRoutine:
  ERROR ERR
    .
    .
```

### ■ノート

▶エラー・コード1〜76は、エラー・メッセージを表示する。

▶参照ステートメント：ON　ERROR、RESUME

▶参照関数：ERR、ERL

# EXIT

## 関数、ループ、プロシージャからの強制脱出

■機能

DEF FN 関数、DO...LOOP、FOR...NEXT、FUNC TION、SUB から抜け出る

■書式

**EXIT {DEF | DO | FOR | FUNCTION | SUB}**

■引数

| 引数 | 機能 |
|---|---|
| DEF | DEF FN による関数の定義から抜ける。 |
| DO | DO...LOOP から抜ける。 |
| FOR | FOR...NEXT から抜ける。 |
| FUNCTION | FUNCTION プロシージャの定義から抜ける。 |
| SUB | SUB プロシージャの定義から抜ける。 |

■文例

```
OPEN "Fruit.Dat" FOR RANDOM AS 1 LEN = 12
FIELD #1, 10 AS nameFld$, 2 AS salesFld$
recNum = 1
DO
  INPUT ; " Fruit (<cr> for EXIT) = ", itName$
  IF itName$ = "" THEN
    EXIT DO
  END IF
  INPUT " , amount = ", amount
  LSET nameFld$ = itName$
  RSET salesFld$ = MKI$(amount)
  PUT #1, recNum
  recNum = recNum + 1
LOOP
CLOSE 1
END
```



■ノート

▶ EXIT DOとEXIT FORは、これを含む一番内側のループを抜け出る。要するに、ループをネスティングしているときは、一つ上のレベルに抜けることになる。

▶参照ステートメント

DEF FN、DO...LOOP、FOR...NEXT、FUNCTION、SUB

# FIELD

## ランダムアクセス・ファイル・バッファの定義

■機能

ランダムアクセス・ファイルのバッファにおける変数を定義しメモリーを割り当てる

■書式

**FIELD　[#] ファイル番号，フィールド幅　AS　文字列変数**
**[，フィールド幅　AS　文字列変数 ...]**

■引数

| 引数 | 機　能 |
|---|---|
| ファイル番号 | OPEN ステートメントでオープンしたときのファイル番号を指定する。 |
| フィールド幅 | フィールド・バッファを構成する要素のバイト数を定義する。 |
| 文字列変数 | フィールド・バッファを構成する要素の名前を設定する。 |

FIELD

■文例①

```
OPEN "Fruit.Dat" FOR RANDOM AS 1 LEN = 12
FIELD #1, 10 AS nameFld$, 2 AS salesFld$
recNum = 1
DO
  INPUT ; " Fruit (<cr> for EXIT) = ", itName$
  IF itName$ = "" THEN
    EXIT DO
  END IF
  INPUT " , amount = ", amount
  LSET nameFld$ = itName$
  RSET salesFld$ = MKI$(amount)
  PUT #1, recNum
  recNum = recNum + 1
LOOP
CLOSE 1
END
```

■文例②

```
OPEN "Fruit.Dat" FOR RANDOM AS 1 LEN = 12
FIELD #1, 10 AS nameFld$, 2 AS salesFld$
recNum = 1
PRINT " N A M E     amount "
PRINT "--------------------"
DO
  GET #1, recNum
  IF EOF(1) THEN
    EXIT DO
  END IF
  itName$ = nameFld$
  amount = CVI(salesFld$)
  PRINT itName$; " = "; USING "##,###"; amount
  recNum = recNum + 1
LOOP
CLOSE 1
END
```

## ■ノート

▶ FIELDバッファは、文字列として取り扱わなければならない。

▶ FIELDバッファに書き込むには、LSET、RSETステートメントを使う。

▶ FIELDバッファに数値を書き込むには、MKI$、MKL$、NKS$、MKD$関数を使って文字列にしてから行なう。

▶ FIELDバッファの文字列になっている数値データは、CVI、CVL、CVS、CVD関数を使って数値にしてから計算などに使う。

▶参照ステートメント

OPEN、GET #、PUT #、CLOSE、MKI$、MKL$、NKS$、MKD$、CVI、CVL、CVS、CVD、LSET、RSET

FIELD

# FILES

## ファイル名の表示

■機能

ディスク上のファイル名を表示する

■書式

FILES ［ファイル名］

■オプション

| オプション | 機　能 |
|---|---|
| ファイル名 | 表示したいファイルまたはディレクトリのパスを指定する。 |

■文例

```
FILES "BIN¥"
```

<実行結果>

```
A:¥QB
           .    <DIR>        ..   <DIR> QB       .EXE  QB        .HLP
QB         .INI     QB         .BI      MOUSE    .COM  BC        .EXE
LIB        .EXE     LINK       .EXE     BRUN42A  .EXE  CODEVIEW.DOC
CV         .EXE     CVPACK     .EXE     CV       .HLP
```

■ノート

▶ファイル名の一部にワイルド・カードが使える。

▶ファイル名を省略すると、カレント・ディレクトリのファイル名を表示する。

# FOR...NEXT

## 繰り返し実行



### 機能

指定した回数だけ一群のステートメントを繰り返し実行する

### 書式

**FOR カウンターj ＝ 初期値 TO 終了値 ［STEP 増分］**

⋮

**NEXT ［カウンターi［, カウンターj...］］**

### 引数とオプション

| 引数 | 機能 |
|---|---|
| カウンターj | ループの回数を数えるために数値変数である。 |
| 初期値 | ループを開始するときに、カウンターjに代入される値を指定する。 |
| 終了値 | カウンターjの最後の値、すなわちテストに使う値を指定する。 |
| STEP 増分 | 1回ループするごとにカウンターjに加える値を指定する。 |

### 文例①

```
FOR n = 1 TO 12
  print month(n)
NEXT n
```

■文例②

```
n = 1
DO WHILE n <= 12
  print month(n)
  n = n + 1
LOOP
```

■ノート

▶ STEP 1のときは、"STEP 増分"を省略できる。

▶増分は、整数でなくてもよい。

▶増分は、負であってもよい。このときは、初期値>=終了値でなければならない。

▶増分が0以上でかつ初期値>終了値ならば、ループ内は一度も実行されない。

▶ NEXT に続くカウンターi、j の部分では、より内側のループの変数を左側に書く。

▶ FOR...NEXT ステートメントによるアルゴリズムは、DO...LOOP ステートメントを使っても可能である(文例②)。

▶論理式のテスト結果に関係なく、ループから抜け出るには、EXIT FOR ステートメントを使う。

▶参照ステートメント:DO、WHILE

# FUNCTION

## FUNCTIONプロシージャの定義

■機能

FUNCTIONプロシージャを定義する。Quick BASICの組み込み関数と同様に、式の中で使う。

■書式

```
FUNCTION プロシージャ名 [(引数リスト)] [STATIC]
    :
    プロシージャ名 = 式
    :
END FUNCTION
```

■引数とオプション

| 引数とオプション | 機　　能 |
|---|---|
| プロシージャ名 | FUNCTIONプロシージャ名を40文字以内で指定する。返す値の型宣言を最後につけるかの型宣言を行なう。 |

FUNCTION

| | |
|---|---|
| 引数リスト | SUBプロシージャがコール側からのパラメーターを受け取るための引数リストである。<br>この引数は、<br>変数　[()]［AS　タイプ］<br>↑　↑　↑<br>SUB内で使う変数名　配列用（省略可）　つぎのどれかで指定する<br>・INTEGER(整数)<br>・LONG()<br>・SINGLE()<br>・DOUBLE()<br>・STRING()<br>（固定長文字列は不可）<br>・ユーザー定義タイプ<br>からなり、複数あるときはカンマで区切る。 |
| 式 | そのFUNCTIONが返す値である。 |
| STATIC | これを指定すると、そのローカル変数は保たれる。 |

■文例①

```
DECLARE FUNCTION binarySearch%(anArray(), target)
DIM userNo(1 TO 100)
  .
theNo = 42
isStat% = binarySearch%(userNo(), theNo)
  .
  .
```

■文例②

```
'　配列データを使ったバイナリー・サーチ
FUNCTION  binarySearch% (theArray(), keyData)
  low = LBOUND(theArray)
  high = UBOUND(theArray)
  WHILE (low <= high)
    mid = INT((low + high) / 2)
    IF (keyData < theArray(mid)) THEN
      high = mid - 1
    ELSEIF (keyData > theArray(mid)) THEN
      low = mid + 1
    ELSE
      binarySearch% = mid
      EXIT FUNCTION
    END IF
  WEND
  binarySearch% = -1
END FUNCTION
```

■ノート

▶ FUNCTION プロシージャが返す値は、その型宣言と同じでなければならない。

▶コール側の引数の数およびタイプと同じでなければならない。

▶引数リストにおける変数の名前は、コール側の変数名と同じである必要はない。

▶引数は、参照渡し(call by reference)されるので、プロシージャ内での書換えが可能である。

▶引数リストにおける変数は、FUNCTION 内においてローカルであるのでコール側に戻るとき自動的に消滅する。

▶引数リストの中で配列変数を使うときは、その変数名に続けて()のみを付ける。

▶プロシージャ名 = 式 を省略すると、数値型ならば0(ゼロ)、文字型ならばヌルを返す。

▶ END FUNCTION 以前でコール側に戻るには、EXIT FUNCTION ステートメントを使う。

# GET [ファイル]

## ランダムアクセス・ファイル/バイナリー・ファイルからの読み込み

### ■機能

ランダムアクセス・ファイルまたはバイナリー・ファイルからデータを読み込む

### ■書式

**GET [#] ファイル番号 [, レコード番号 [, 変数]]**

### ■引数

| 引数 | 機能 |
|---|---|
| ファイル番号 | OPEN ステートメントでオープンしたときのファイル番号を指定する。 |
| レコード番号 | ランダムアクセス・ファイルのときは、次に読み込むレコードの番号(1～2147483647)を指定する。<br>バイナリー・ファイルのときは、次に読み込むバイトの開始位置(1～ファイル・サイズ)を指定する。 |
| 変数 | 読み込むデータを格納するための変数を指定する。 |

### ■文例①

```
TYPE FoodRecType
   itName AS STRING * 10
   amount AS INTEGER
   END TYPE
DIM fruit AS FoodRecType

OPEN "Fruit.Dat" FOR RANDOM AS 1 LEN = LEN(fruit)
recNum = 1
PRINT " N A M E       amount "
PRINT "----------------------"
```

```
DO
  GET #1, recNum, fruit
  IF EOF(1) THEN
    EXIT DO
  END IF
  PRINT fruit.itName; " = "; USING "##,###"; fruit.amount
  recNum = recNum + 1
LOOP
CLOSE 1
END
```

■文例②

```
' ファイルのコピー
DIM oneByte AS STRING * 1
' enter file names ...
INPUT "Source file name = ", sourceFile$
INPUT "Destination file name = ", destFile$
' open files ...
OPEN sourceFile$ FOR BINARY AS 1
OPEN destFile$ FOR BINARY AS 2
bytePos = 1
DO
  GET #1, bytePos, oneByte
  IF EOF(1) THEN
    EXIT DO
  END IF
  PUT #2, bytePos, oneByte
  bytePos = bytePos + 1
LOOP
CLOSE 1, 2
END
```

■ノート

▶"変数"にレコード型変数を使うときは、その宣言名だけでよい。

▶ FIELDステートメントによってバッファを割り当てるときは、"変数"の部分は省く(FIELDステートメントを参照)。

▶バイナリー・ファイルからは、その変数のサイズに等しいバイト数だけ読み込まれる。

▶参照ステートメント：OPEN、PUT #、FIELD、CLOSE

# GET［グラフィックス］

## グラフィック・イメージの保存

■機能

スクリーンのグラフィック・イメージを配列変数に保存する

■書式

**GET ［STEP］(x1, y1) － ［STEP］(x2, y2),**
**配列名［(インデックス)］**

■引数とオプション

| 引数とオプション | 機　能 |
|---|---|
| STEP | 現在のグラフィック・カーソルの位置から相対的にx, yの座標を決めたいときに指定する。 |
| (x1, y1)<br>(x2, y2) | 保存したい範囲の対角座標を指定する。 |
| 配列名 | 保存先の数値型配列変数名を指定する。 |
| インデックス | 保存するイメージをどの配列要素から始めるかを指定する。 |

## ■文例

```
READ x1, y1, x2, y2
DATA 1, 1, 100, 100
bytes = 4 + 4 * INT(((x2 - x1 + 1) + 7) / 8) * (y2 - y1 + 1)
DIM imageSample(bytes / 2) AS INTEGER
SCREEN 88, , 0, 1
LINE (x1, y1)-(x2, y2), , BF
GET (x1, y1)-(x2, y2), imageSample
PUT (200, 200), imageSample
PCOPY 0, 1
  .
  .
  .
```

## ■ノート

▶保存に必要な配列のバイト・サイズは、つぎの式で求めます。

4+4＊INT(((x2-x1+1)+7)/8)＊(y2-y1+1)

▶保存に必要な配列の上限は、上の式で求めたサイズを満たすように設定しなければならない。

| 型 | 1要素のバイト数 |
|---|---|
| 整数 | 2バイト |
| 長整数 | 4バイト |

▶参照ステートメント：PUT、BSAVE、BLOAD

# GOSUB

## サブルーチンへの分岐

■機能

サブルーチンへ分岐する

■書式

**GOSUB　{行番号｜行ラベル}**

■引数

| 引数 | 機　　能 |
|---|---|
| 行番号 | 分岐したいサブルーチンの行番号またはラベルを指定する。<br>行番号は、0から65529の範囲でなければならない。 |
| 行ラベル | 行ラベルについてのルール<br>・文字で始めなければならない<br>・":"で終わらなければならない<br>・40文字以内でなければならない<br>・大文字と小文字の区別はない |

■文例

```
IF yesNo$ = "YES" THEN
  GOSUB GetNameRoutine
ELSE
  GOSUB GetAddrRoutine
END IF
   .
   .
END
GetNameRoutine:          ' サブルーチン
   .
   .
  RETURN
GetAddrRoutine:          ' サブルーチン
   .
   .
  RETURN
```

■ノート

- ▶ GOSUBステートメントにおける行ラベル名の末尾には、":"をつけない。
- ▶サブルーチンから戻るには、RETURNステートメントを使う。
- ▶他のモジュールやプロシージャのサブルーチンへは分岐できない。
- ▶参照ステートメント：RETURN

# GOTO

## 強制分岐



■機能

特定の行に無条件で分岐する

■書式

GOTO {行番号|行ラベル}

■引数

| 引数 | 機能 |
|---|---|
| 行番号 | 強制的に分岐したい行番号またはラベルを指定する。<br>行番号は、0から65529の範囲でなければならない。 |
| 行ラベル | 行ラベルについてのルール<br>・文字で始めなければならない<br>・":"で終わらなければならない<br>・40文字以内でなければならない<br>・大文字と小文字の区別はない |

■文例

```
IF yesNo$ = "YES" THEN
  GOTO GetNameRoutine
ELSE
  GOTO GetAddrRoutine
END IF
  .
  .
  .
GetNameRoutine:
  .
  END
GetAddrRoutine:
  .
  END
```

■ノート

▶ GOTOステートメントにおける行ラベル名の末尾には、":"をつけない。

▶他のモジュールやプロシージャへGOTOステートメントを使って分岐はできない。

GOTO

# IF

**分岐**

## 機能

論理式の評価結果に基づいて分岐し実行する。

書式①は、1行で書くIFステートメントであり、書式②は、ブロック形式のIFステートメントである。

## 書式

**① IF　論理式　THEN　thenパート　[ELSE　elseパート]**

**② IF　論理式1　THEN**
**　　　　ステートメント・ブロック1**
**　　　　[ELSEIF　論理式2　THEN**
**　　　　[ステートメント・ブロック2]**
**　　　⋮**
**　　　　[ELSE**
**　　　　ステートメント・ブロックn]]**
**　END　IF**

## 引数とオプション(書式①)

| 引数 | 機　能 |
|---|---|
| 論理式 | テストすべき式を指定。その結果は、真または偽になる。 |
| thenパート | 論理式が真のとき実行させたいステートメントを書く。 |
| elseパート | 論理式が偽のとき実行させたいステートメントを書く。 |

## 文例①

```
IF isResponse$ = "Y" THEN PRINT "Yes" ELSE PRINT "No "
```

■ノート

▶ ELSE 以下の部分は省略できる。

▶ "then パート" と "else パート" には複数のステートメントが書ける。ここでは、GOTO で分岐する行の指定に行番号を使うときは GOTO を省略できる。

■引数とオプション(書式②)

| 引数とオプション | 機能 |
| --- | --- |
| 論理式1〜m | テストすべき式を指定。その結果は、真または偽になる。 |
| ステートメント・ブロック1〜m | 論理式1〜mが真のときに実行させたいステートメントを書く。 |
| ステートメント・ブロックn | 論理式1〜mのすべてが偽のときに実行させたいステートメントを書く。 |

■文例②

```
SUB binarySearch (isStat, theArray(), keyData)
  low = LBOUND(theArray)
  high = UBOUND(theArray)
  WHILE (low <= high)
    mid = INT((low + high) / 2)
    IF (keyData < theArray(mid)) THEN
      high = mid - 1
    ELSEIF (keyData > theArray(mid)) THEN
      low = mid + 1
    ELSE
      isStat = mid
      EXIT SUB
    END IF
  WEND
  isStat = -1
END SUB
```

■ノート

▶構造化プログラミングには、書式②の方が適している。

▶ "ELSE" 以下の部分は省略できる。

▶ "END　IF" の部分は省略できない。

▶参照ステートメント：SELECT　CASE

IF

# INPUT

## キーボードからの入力

INPUT

■機能

キーボードからの入力を行なう

■書式

**INPUT　[；]［"プロンプト文字列"｛；｜，｝］ 変数リスト**

■引数とオプション

| 引数とオプション | 機　能 |
|---|---|
| ； | INPUTのすぐ後にセミコロンを書くと、入力後もカーソルは同じ行に留まる。 |
| プロンプト文字列 | 入力用プロンプトを表示したいときに指定する。<br>プロンプトの後にセミコロンを書くと"？"マークを表示し、カンマを書くと"？"マークを表示しない。 |
| 変数リスト | 入力したい変数を指定する。 |

■文例①

```
INPUT "Car name = "; aName$
PRINT "Price = "; : INPUT amount
```

■文例②

```
TYPE productRecType
  itName AS STRING * 10
  amount AS INTEGER
  END TYPE
DIM car AS productRecType
  .
  .
INPUT ; " Car name = ", car.itName
INPUT " and  amount = ", car.amount
  .
  .
  .
```

■ノート

▶入力は、改行キー(リターン・キー)を押すことによって確定する。

▶変数を2つ以上指定したときは、入力データをカンマで区切って入力する。

▶参照ステートメント：LINE INPUT

INPUT

# INPUT #［ファイル］

## シーケンシャル・ファイルからの読み込み

### ■機能

シーケンシャル・ファイルやデバイスからデータを読み込む

### ■書式

**INPUT #ファイル番号，変数リスト**

### ■引数

| 引数 | 機　能 |
|---|---|
| #ファイル番号 | OPENステートメントでオープンしたときのファイル番号を指定する。 |
| 変数リスト | 読み込むデータを格納するための変数を指定する。 |

### ■文例

```
TYPE FoodRecType
  itName AS STRING * 10
  amount AS INTEGER
  END TYPE
DIM fruit AS FoodRecType

OPEN "Fruit.Dat" FOR INPUT AS 1
PRINT " N A M E      amount "
PRINT "---------------------"
DO WHILE NOT EOF(1)
  INPUT #1, fruit.itName, fruit.amount
  PRINT fruit.itName; " = "; USING "##,###"; fruit.amount
LOOP
CLOSE 1
END
```

### ■ノート

▶フィールドの区切り(”，”)ごとに変数を読み込む。

▶参照ステートメント：INPUT #

# IOCTL

## デバイス・ドライバーへの制御データの送出

### ■機能

デバイス・ドライバーへ制御データを送る

### ■書式

**IOCTL ［#］ファイル番号，文字列**

### ■引数

| 引数 | 機　能 |
|---|---|
| ファイル番号 | デバイス・ドライバーをオープンしたときのファイル番号を指定する。 |
| 文字列 | 制御に必要なデータを文字列にして指定する。 |

### ■文例

```
OPEN "¥Driver¥HardCopy" OUTPUT AS 1
IOCTL 1, "XFD"
```

### ■ノート

▶参照関数：IOCTL$

# KEY

## ファンクション・キーの割り当てと表示

■機能

ファンクション・キーへの割り当てを行なう。キー内容のスクリーン表示も行なえる。

■書式

**KEY {n, 文字式 | LIST | ON | OFF}**

■引数

| 引数 | 機能 |
|---|---|
| n, 文字式 | ファンクション・キーへの割り当てを行なう。<br>nとして1～10を指定でき、文字式は15文字までである。 |
| LIST | ファンクション・キーの一覧を表示する。 |
| ON | ファンクション・キーの内容をスクリーンの最下行に表示する。 |
| OFF | ファンクション・キーの内容を表示しない。 |

## ■文例

```
CLS
DIM menu$(1 TO 10)
FOR i = 1 TO 10
  READ menu$(i)
  KEY i, menu$(i)
NEXT i
KEY ON
   .
   .
   .
menuResource:
DATA New, Open, Save, Close, Quit
DATA Enter, Edit, Find, Print, Undo
```

**<表示画面>**

```
1New 2Open 3Save 4Close 5Quit 6Enter 7Edit 8Find 9Print 10Undo
```

## ■ノート

▶文字式を使って、ユーザー定義キーについてのマスク処理とスキャンを行なえる。

▶参照ステートメント：KEY(n)

KEY

# KEY(n)

## キーの割り込み



### ■機能

任意のキーの割り込みを開始または終了する

### ■書式

**KEY(n)　{ON | OFF | STOP}**

### ■引数とオプション

| 引数 | 機　能 |
|---|---|
| (n) | キーの番号を指定する。 |
| ON | キー・イベント・トラッピングを有効にする。<br>有効であるあいだ、ON TIMER(n) GOSUB ステートメントで設定した秒数が経過したかどうかチェックし、経過したときは、そのイベント・トラッピングのサブルーチンを実行する。 |
| OFF<br>STOP | キー・イベント・トラッピングを無効にする。<br>OFF は、そのイベントを記憶しないが、STOP は、記憶しているのでトラッピングできるようになると ON KEY ルーチンを実行する。 |

### ■文例

```
KEY 15, CHR$(&H01+&H10)+CHR$(&H01)      ' ﾏｽｸ･ｺｰﾄﾞとｽｷｬﾝ･ｺｰﾄﾞ
KEY(15) ON
ON KEY(15) GOSUB ShiftCtrl1
   .
   .
   .
```

## ■ノート

▶キーの番号は、つぎのようになっている。

| キー | 番号 |
|---|---|
| ファンクション・キー | 1〜10 |
| [↑] キー | 11 |
| [←] キー | 12 |
| [→] キー | 13 |
| [↓] キー | 14 |
| ユーザー定義キー | 15〜25 |

▶ベースとなるキー([SHIFT]、[CAPS]、[カナ]、[GRPH]、[CTRL])などが押されているかどうかは、つぎのマスク・コードにて識別できる。2つ以上のときは、それらのコードを加えればよい。

| キー | コード |
|---|---|
| マスクなし | &H00 |
| [SHIFT] キー | &H01 |
| [CAPS] キー | &H02 |
| カナ キー | &H04 |
| [GRPH] キー | &H08 |
| [CTRL] キー | &H10 |

▶キー・スキャン・コードについて付録を参照してください。

▶参照ステートメント：KEY、ON event GOSUB

KEY(n)

# KILL

## ファイルの消去



■機能

ディスク上のファイルを消去する

■書式

**KILL　ファイル名**

■引数

| 引数 | 機　能 |
|---|---|
| ファイル名 | 削除したいファイル名を指定する。 |

■文例

```
KILL "SETTING.BAK"
```

■ノート

▶ MS-DOSのDELおよびERASEコマンドと同じ機能をもっている。

▶ ファイル名の指定にワイルド・カードが使える。

# KMID$

## 文字列の置き換え

### ■機能

文字列変数の一部を他の文字列で置き換える。1バイト文字と2バイト文字を混在できる。

### ■書式

**KMID$(文字列変数, 開始位置 [, 文字数]) = 文字式**

### ■引数とオプション

| 引数とオプション | 機　能 |
|---|---|
| 文字列変数 | 置き換えの対象となる文字列変数を指定する。 |
| 開始位置 | 置き換えの対象となる部分文字列の最初に位置を指定する。 |
| 文字数 | 置き換えの対象となる部分文字列の長さをそのバイト数で指定する。 |
| 文字式 | 新しく置き換える文字列を指定する。 |

### ■文例

```
'                ....*.....1....*....2....
oldString$ = "ＱｕｉｃｋＢＡＳＩＣ(version ４．２０)"
MID$(oldString$, 6, 5) = "BASIC"
PRINT oldString$    ' "ＱｕｉｃｋBASIC(version ４．２０)"になる.
```

### ■ノート

▶"文字数"を省略すると、"文字式"の長さを指定したことと同じになる。

▶"文字数"が"文字式"の長さよりも短いならば、その文字数だけ置き換わる。

▶"文字数"が"文字式"の長さよりも長いならば、その文字式の長さだけ置き換わる。

▶参照ステートメント：MID$

# LET

## 変数への代入

### ■機能

変数に値を代入する

### ■書式

［LET］　変数 ＝ 式

### ■引数とオプション

| 引数 | 機　　能 |
|---|---|
| 変数 | 代入したい変数を指定する。 |
| 式 | 変数に代入したい定数、変数、または式を指定する。 |

### ■文例①

```
DIM month(12)
corpName$ = "株式会社　ＡＢＣ"
month(5) = 42000
```

■文例②

```
TYPE aPersonalRecordType      'ユーザー定義型
  firstName AS STRING * 10
  secondName AS STRING * 10
  age AS INTEGER
  phone AS STRING * 12
END TYPE

DIM user AS personalRecordType

LET user.firstName = "Bill"
LET user.secondName = "Evans"
LET user.age = 42
LET user.phone = "03(291)1257"
```

■ノート

▶ LET は省略できる。

▶ユーザー定義型の変数に代入するとき、文例②の変数名は、レコード型変数名とその型の要素名(firstName, age など)とを". "でつなぐ。

　レコード型宣言名. 要素名

▶同じタイプのユーザー定義型の変数全体を代入するときは、要素名を省略できる。

LET

# LINE

## 線・ボックスの描図

■機能

スクリーンの線またはボックスを描く

■書式

**LINE** [[**STEP**] (x1, y1)] - [STEP] (x2, y2)
[, [カラー] [, [**B** | **BF**] [, スタイル]]]

■引数とオプション

| 引数とオプション | 機　能 |
|---|---|
| STEP | 現在のグラフィック・カーソルの位置から相対的に x、y の座標を決めたいときに指定する。 |
| (x1, y1)<br>(x2, y2) | 線の両端の座標またはボックスの対角座標を指定する。 |
| カラー | 線またはボックスを描くときのカラーを指定する。 |
| B<br>BF | B は、座標の指定がボックスのものであることを指定する。<br>BF は、ボックスでかつその内部をカラーで塗りつぶすことを指定する。 |
| スタイル | 線のスタイルを16ビット(2バイト)のビットパターンで指定する。 |

■文例

```
LINE (50, 100)-(200, 300), , B,&H1010
```

■ノート

▶座標の指定は、(x1, y1)と(x2, y2)のどちらに行なっても同じである。

▶スタイルのビット・パターンは、ビットが1ならばそのピクセルを表示し、0ならば表示しない。

▶参照ステートメント：SCREEN

LINE

# LINE INPUT

## キーボードからの入力



■機能

区切り文字の有無に関係なく、キーボードから入力されるもの全体を1つのものとして文字列変数に入力する

■書式

**LINE INPUT [;][”プロンプト文字列”;] 文字列変数**

■引数とオプション

| 引数とオプション | 機能 |
|---|---|
| ; | INPUTのすぐ後にセミコロンを書くと、入力後もカーソルは同じ行に留まる。 |
| プロンプト文字列 | 入力用プロンプトを表示したいときに指定する。 |
| 文字列変数 | 入力したい文字列変数を指定する。 |

■文例

```
LINE INPUT "Your address ? " anAddress$
```

■ノート

▶”?”マークは表示しない。

▶最大255文字まで入力可能である。

▶参照ステートメント：INPUT

# LINE INPUT #

## シーケンシャル・ファイルからの読み込み

■機能

区切り文字の有無に関係なく、シーケンシャル・ファイルからデータを文字列変数に読み込む

■書式

**LINE INPUT #ファイル番号，文字列変数**

■引数

| 引数 | 機能 |
|---|---|
| #ファイル番号 | OPENステートメントでオープンしたときのファイル番号を指定する。 |
| 文字列変数 | 読み込むデータを格納する文字列変数を指定する。 |

■文例

```
OPEN "Fruit.Dat" FOR INPUT AS 1
PRINT " N A M E     amount "
PRINT "----------------------"
DO WHILE NOT EOF(1)
  LINE INPUT #1, oneRecord$
  PRINT oneRecord$
LOOP
CLOSE 1
END
```

■ノート

▶フィールドの区切り(",")に関係なく、レコードの区切り(&H0D, &H0A)までを1つの変数に読み込む。

▶参照ステートメント：INPUT #

# LOCATE

## カーソル制御

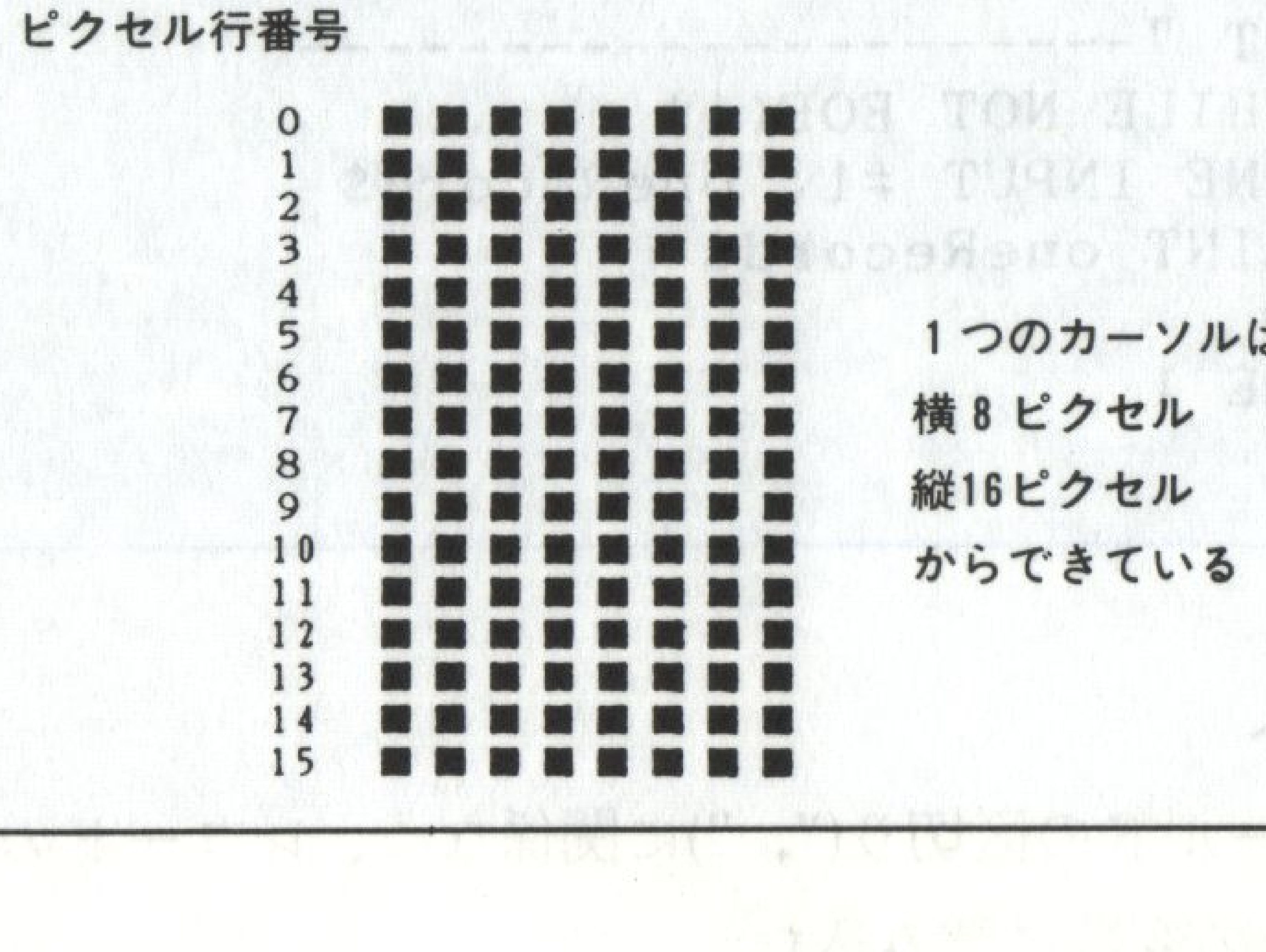
**■機能**

テキスト・カーソルの移動を行ない、表示の有無を決め、および形状を設定する。

**■書式**

**LOCATE** ［行］［,［桁］
［,［カーソル］［,［スタート］　［, ストップ］］］］

**■オプション**

<table>
<tr><th>オプション</th><th>機　能</th></tr>
<tr><td>行</td><td>移動した行(1〜25)を指定する。</td></tr>
<tr><td>桁</td><td>移動した桁(1〜80)を指定する。</td></tr>
<tr><td>カーソル</td><td>表示の有無を指定する。1ならば表示し、0ならば表示しない。</td></tr>
<tr><td>スタート<br>ストップ</td><td>カーソルの形状をピクセル行の範囲で設定する。<br>
</td></tr>
</table>

■文例

```
LOCATE 20, 1
INPUT "Your name ? ", aName$
```

```
LOCATE 20, 1, 1,14, 15
INPUT "Your name ? ", aName$
```

■ノート

▶各オプションを省略するとそれぞれそのときの設定になる。

▶"ストップ"を省略すると"スタート"と同じピクセル行を指定したことになる。

▶"ストップ"が"スタート"よりも小さいならば、カーソルの形状は反転する。

# LOCK

## ファイル・アクセスのロック

■機能

ネットワークのもとでオープンされているファイルに対するアクセスをロックする

■書式

**LOCK　[#] ファイル番号**
**[, {レコード | [開始位置]　TO　終了位置}]**

■引数とオプション

| 引数とオプション | 機　能 |
|---|---|
| ファイル番号 | ロックしたいファイルの番号を指定する。 |
| レコード | ロックしたいレコードまたはバイト位置を指定する。 |
| 開始位置<br>終了位置 | ロックしたい範囲をレコードまたはバイト位置で指定する。 |

■文例

```
LOCK #1, 1 TO 100
LOCK #2
```

■ノート

▶ランダムアクセス・ファイルでは、レコード番号で行なう。
▶バイナリー・ファイルでは、バイト位置で行なう。
▶シーケンシャル・ファイルでは、ファイル全体がロックされる。
▶ファイル番号だけを指定すると、ファイル全体をロックする。
▶"TO　終了位置"だけを指定すると、ファイルの先頭からそこまでをロックする。
▶参照ステートメント：UNLOCK

# LPRINT

## プリンター印字



■機能

プリンターにデータを印字する

■書式

LPRINT ［USING 書式制御文字列；］
［式リスト］［{, | ;}］

■引数とオプション

| 引数とオプション | 機　能 |
|---|---|
| USING | 書式制御文字列を使うときに指定する。 |
| 書式制御文字列 | 式の値の表示形式や位置を制御したいときに使う。<br>●文字列の表示 |

| 制御文字 | 機　能 |
|---|---|
| @ | その位置からすべてを印字する。 |
| & | 2つの&の範囲に印字する。 |
| ! | 最初に1文字のみを印字する。 |

●数値の印字

| 制御文字 | 機　能 |
|---|---|
| # | 右詰めで印字する。 |
| , | カンマを印字する。 |
| ¥¥ | 先頭に¥を付ける。 |
| ** | 空白部分を＊で埋める。 |
| . | 小数点を印字する。 |
| + | 符号を先頭または末尾に印字する。 |
| − | ーを先頭に印字する。 |
| ^ | 指数形式で印字する。 |
| _ | 書式制御文字を文字として印字する。 |

| | |
|---|---|
| 式リスト | 印字したい値を指定する。<br>式は、","または";"で区切る。 |

■文例①

```
LPRINT "Quick","BASIC"
LPRINT "Quick";"BASIC"
LPRINT
LPRINT "Quick";
LPRINT "BASIC"
```

<印字結果>

```
Quick              BASIC
QuickBASIC

QuickBASIC
```

■文例②

```
LPRINT USING "@BASIC"; "Quick"
LPRINT USING "& &BASIC"; "Quick"
LPRINT USING "!BASIC"; "Quick"
LPRINT USING "######### 円"; 4200
LPRINT USING "##,###,### 円"; 4200
LPRINT USING "¥¥,###,###"; 4200
LPRINT USING "**,###,###"; 4200
LPRINT USING "QucikBASIC(#.##); 4.2
LPRINT USING "+#.##"; -4.2
LPRINT USING "#.##-"; -4.2
LPRINT USING "#.#^^^^"; 4.2
LPRINT USING "######,#"; 420000000
LPRINT USING "#.## "; 3.1; 5.7; .38
```

<印字結果>

```
QuickBASIC
QuiBASIC
QBASIC
     4200 円
   4,200 円
  ¥4,200
*****4,200
QucikBASIC(4.20)
-4.20
4.20-
0.4E+01
%420,000,000
3.10 5.70 0.38
```

■ノート

▶引数およびオプションを省略すると改行する。

▶ステートメントの最後に","または";"を付けると改行しないで次の表示を行なう(USINGでは";"だけ使える)。

▶ USINGを使わない式リスト中での式は、等間隔な印字を行なうには","で区切り、または続けて印字するには";"で区切らなければならない。

▶ USINGを使った式リスト中での式は、";"で区切らなければならない。

▶","は、"#"の中に1つあれば3桁ごとに区切ることができる。

▶オバーフローすると先頭に"%"が印字される。

▶ 1つの書式制御文字列では、2つ以上の式を同じ書式で印字できる。

▶書式制御文字列には、文字列変数が使える。

LPRINT

# LSET

## 文字列変数への左詰め代入

### ■機能

文字列変数に左詰めで代入する。

### ■書式

**LSET 文字列変数 = 文字式**

### ■引数

| 引数 | 機能 |
|---|---|
| 文字列変数 | 文字列変数やランダムアクセス・バッファの変数を指定する。 |
| 文字式 | 左詰めにしたい値の文字式を指定する。 |

### ■文例

```
OPEN "Fruit.Dat" FOR RANDOM AS 1 LEN = 12
FIELD #1, 10 AS nameFld$, 2 AS salesFld$
recNum = 1
DO
  INPUT ; " Fruit (<cr> for EXIT) = ", itName$
  IF itName$ = "" THEN
    EXIT DO
  END IF
  INPUT " , amount = ", amount%
  LSET nameFld$ = itName$
  RSET salesFld$ = MKI$(amount%)
  PUT #1, recNum
  recNum = recNum + 1
LOOP
CLOSE 1
END
```

### ■ノート

▶ FIELDバッファに数値を書き込むには、MKI$、MKL$、NKS$、MKD$関数を使って文字列にしてから行なう。

# MID$

## 文字列の置き換え

### ■機能

文字列変数の一部を他の文字列で置き換える

### ■書式

**MID$(文字列変数, 開始位置 [, 文字数]) = 文字式**

### ■引数とオプション

| 引数とオプション | 機　能 |
|---|---|
| 文字列変数 | 置き換えの対象となる文字列変数を指定する。 |
| 開始位置 | 置き換えの対象となる部分文字列の最初に位置を指定する。 |
| 文字数 | 置き換えの対象となる部分文字列の長さをそのバイト数で指定する。 |
| 文字式 | 新しく置き換える文字列を指定する。 |

### ■文例

```
'                ····*····1····*····2····
oldString$ = "QuickBASIC version 4.20"
MID$(oldString$, 12, 7) = "ﾊﾞｰｼﾞｮﾝ"
PRINT oldString$    ' "QuickBASIC ﾊﾞｰｼﾞｮﾝ 4.20"になる
```

### ■ノート

▶”文字数”を省略すると、”文字式”の長さを指定したことと同じである。

▶”文字数”が”文字式”の長さよりも短いならば、その文字数だけ置き換わる。

▶”文字数”が”文字式”の長さよりも長いならば、その文字式の長さだけ置き換わる。

▶参照ステートメント：KMID$

# MKDIR

## 新しいサブ・ディレクトリの作成

### ■機能

ディスク上に新しいサブ・ディレクトリを作る

### ■書式

**MKDIR　パス名**

### ■引数

| 引数 | 機　能 |
|---|---|
| パス名 | 作成したいパス名を文字式で指定する。 |

### ■文例

```
MKDIR "MAILING"
thePath$ = "B:¥MODULES¥DOSCALL"
MKDIR thePath$
```

### ■ノート

▶ディレクトリの変更は、CHDIRステートメントを使う。

▶ディレクトリの削除は、RMDIRステートメントを使う。

# NAME

## ファイル名の変更/ファイルの移動

### ■機能

パスの指定が同じならば、ディスク上のファイルの名前を変更する。またパスの指定が異なるならば、ファイルの移動を行なう。

### ■書式

**NAME　旧ファイル名　AS　新ファイル名**

### ■引数

| 引数 | 機　能 |
|---|---|
| 旧ファイル名 | 変更する前のファイル名を指定する。 |
| 新ファイル名 | 変更した後のファイル名を指定する。 |

### ■文例

```
NAME "PRINTING.BAS" AS "LISTING.BAS"
```

### ■ノート

▶"旧ファイル名"は、必ず存在していなければならず、また逆に"新ファイル名"と同名のファイルは、すでに存在していてはいけない。

▶"旧ファイル名"と"新ファイル名"の指定で、パス名が同じだと同じディレクトリ内でその名前を変更し、パス名が別のときはそのファイルを移動する。

▶オープン中のファイルについては変更できない(ERR =55)。

▶別のディスク間では行えない(ERR =74)。

▶ファイル名の指定でワイルド・カードは使えない(ERR =36)。

# ON ERROR GOTO

## エラー処理ルーチンの開始行の指定



### ■機能

エラーが発生したときに使うエラー処理ルーチンの開始行を指定する。またエラー処理を行わないようにもできる。

### ■書式

**ON ERROR GOTO {行番号|行ラベル}**

### ■引数

| 引数 | 機能 |
|---|---|
| 行番号<br>行ラベル | エラー処理ルーチンの開始行を行番号または行ラベルで指定する |

### ■文例

```
ON ERROR GOTO ErrorHandleRoutine
   .
   .
ErrorHandleRoutine:
  ERROR ERR
   .
  RETURN
```

### ■ノート

▶行番号として0を指定すると、エラーが発生してもエラー処理を行なわない(プログラムはその時点で止まってしまう)。

▶エラー処理ルーチンは、モジュール・レベルのコードとして書かなければならない。

▶参照ステートメント：ERROR、RESUME

▶参照関数：ERR、ERL

# ON event GOSUB

(Ver.4.5 拡張)

## イベント・トラップ用サブ・ルーチンの開始行の指定

■機能

イベント・トラップの分岐先であるサブ・ルーチンの開始行を指定する。

■書式

ON イベント GOSUB {行番号 | 行ラベル}



■引数

<table>
<tr><th>引数</th><th colspan="2">機能</th></tr>
<tr><td rowspan="6">イベント</td><td colspan="2">イベントの種類を指定する。</td></tr>
<tr><th>種類</th><th>イベント</th></tr>
<tr><td>COM(n)</td><td>通信イベント。nは、通信ポート(1, 2)を指定する。</td></tr>
<tr><td>KEY(n)</td><td>キー・イベント。nは、キーの番号を指定する。</td></tr>
<tr><td>PLAY(n)</td><td>演奏イベント。nは、バックグランドミュージックのキューにおける最小音符数(1～32)を指定する。</td></tr>
<tr><td>TIMER(n)</td><td>タイム・イベント。nは、経過秒数を指定する。</td></tr>
<tr><td>行番号<br>行ラベル</td><td colspan="2">分岐先を行番号または行ラベルで指定する。</td></tr>
</table>

■文例

```
TIMER ON
ON TIMER(5) GOSUB skipToNext
DO
LOOP

skipToNext:
TIMER OFF
  .
  .
  .
```



■ノート

▶参照ステートメント：COM(n)、KEY(n)、TIMER

# ON...GOSUB

## サブ・ルーチンへの分岐

■機能

式の値に基づいてそれぞれのサブ・ルーチンへ分岐する

■書式

**ON 式 GOSUB {行番号リスト|行ラベル・リスト}**

■引数

| 引数 | 機　能 |
|---|---|
| 式 | 分岐したいリスト上の値になる式を設定する。 |
| 行番号リスト<br>または<br>行ラベル・リスト | 分岐したいサブルーチンの行番号またはラベルを指定する。<br>行番号は、0から65529の範囲でなければならない。 |

■文例

```
GetMenuNumber:
  INPUT "Menu No. ? ( 1 - 5 ) ", menuNo
  IF (menuNo >= 1) AND (menuNo <= 5) THEN
    ON menuNo GOSUB Enter, Edit, Delete, Find, Listing
    '  menuNo   =   1    , 2   , 3      , 4   , 5
  ELSE
    PRINT "Bad Menu No !!!"
    GOTO GetMenuNumber
  END IF
   .
   .
Enter:       ' menuNo = 1
   .
   .
```

■ノート

▶式の値は、整数値として評価される。

▶行番号リストまたは行ラベル・リストは、式の値1、2、3...に対応して分岐する。

▶行番号リストまたは行ラベル・リスト中に、式の値に対応するものがなければ、そのON...GOSUBの次のステートメントを実行する。

▶参照ステートメント：ON...GOTO、SELECT

# ON...GOTO

## 行への分岐

■機能

式の値に基づいてそれぞれの行へ強制的に分岐する

■書式

**ON　式　GOTO　{行番号リスト｜行ラベル・リスト}**

■引数とオプション

| 引数 | 機　能 |
|---|---|
| 式 | 分岐したいリスト上の値になる式を設定する。 |
| 行番号リスト<br>行ラベル・リスト | 分岐したい行番号またはラベルを指定する。<br>行番号は、0から65529の範囲でなければならない。 |

■文例

```
GetMenuNumber:
  INPUT "Menu No. ? ( 1 - 5 ) ", menuNo
  IF (menuNo >= 1) AND (menuNo <= 5) THEN
    ON menuNo GOTO Enter, Edit, Delete, Find, Listing
    '  menuNo   =   1    , 2    , 3      , 4    , 5
  ELSE
    PRINT "Bad Menu No !!!"
    GOTO GetMenuNumber
  END IF
    .
    .
    .
Enter:        ' menuNo = 1
    .
    .
  GOTO GetMenuNumber
```

■ノート

▶参照ステートメント：ON...GOSUB、SELECT

# OPEN(ファイル)

## ファイルのオープン



■機能

ディスク上のファイルまたはデバイスをオープンして入出力できるようにする

■書式

**① OPEN ファイル名 [FOR モード]**
**[ACCESS アクセス] [ロック] AS [#] ファイル番号**
**[LEN = レコード・サイズ]**

**② OPEN モード, [#] ファイル番号, ファイル名**
**[, レコード・サイズ]**

■引数とオプション(書式①)

| 引数 | 機能 |
|---|---|
| ファイル名 | オープンしたいファイルの名前を指定する。 |
| モード | つぎの中からファイルのタイプを指定する。 |

| モード | ファイル |
|---|---|
| INPUT | シーケンシャル・ファイルから読み込む。 |
| OUTPUT | シーケンシャル・ファイルへ書き出す。ファイルの先頭から書き出す。 |
| APPEND | シーケンシャル・ファイルへ書き出す。ファイルの終わりに追加する。 |
| RANDOM | ランダム・アクセス・ファイルをオープン。 |
| BINARY | バイナリー・ファイルとしてオープン。 |

<table>
<tr><td>アクセス</td><td>ファイルへのアクセスの仕方を指定する。
<table>
<tr><th>アクセス</th><th>機　能</th></tr>
<tr><td>READ</td><td>読み込みだけを行なう。</td></tr>
<tr><td>WRITE</td><td>書き出しだけを行なう。</td></tr>
</table>
READ WRITE の両方を一度に指定できる。</td></tr>
<tr><td>ロック</td><td>ネットワーク環境でのアクセスを制御する。
<table>
<tr><th>タイプ</th><th>機　能</th></tr>
<tr><td>SHARED</td><td>共有させる。</td></tr>
<tr><td>LOCK READ</td><td>最初にリード・アクセスのあったプロセスのみに以後のリード・アクセスを許す。</td></tr>
<tr><td>LOCK WRITE</td><td>最初にライト・アクセスのあったプロセスのみに以後のライト・アクセスを許す。</td></tr>
<tr><td>LOCK READ WRITE</td><td>最初にリード・またはライト・アクセスのあったプロセスのみに以後のアクセスを許す。</td></tr>
</table>
タイプを省略するとオープンしているときは、他のプロセスからのアクセスを禁止する。</td></tr>
<tr><td>ファイル番号</td><td>オープンしたファイルに割り当てる番号を指定する。オープン済みファイルの指定は、この番号を使う。</td></tr>
</table>

OPEN

| レコード・サイズ | オープンするファイルのタイプのレコード長を指定する。 | |
|---|---|---|
| | ファイル | レコード・サイズ |
| | シーケンシャル | 省略可能<br>デフォルトのバッファ・サイズは、512バイトになっている。 |
| | ランダム | 1レコードのバイト数を指定する。<br>デフォルトのバッファ・サイズは、128バイトになっている。 |
| | バイナリー | 不要 |

OPEN

■文例

```
TYPE FoodRecType
  itName AS STRING * 10
  amount AS INTEGER
  END TYPE
DIM fruit AS FoodRecType

OPEN "Fruit.Dat" FOR RANDOM AS 1 LEN = LEN(fruit)
recNum = 1
PRINT " N A M E      amount "
PRINT "---------------------"
DO
  GET #1, recNum, fruit
  IF EOF(1) THEN
    EXIT DO
  END IF
  PRINT fruit.itName; " = "; USING "##,###"; fruit.amount
  recNum = recNum + 1
LOOP
CLOSE 1
END
```

■ノート

▶モードのデフォルトは、"RANDOM"になっている。

▶未使用のファイル番号は、FREEFILE関数で知ることができる。

▶ランダム・アクセス・ファイルおよびバイナリー・ファイルで"アクセス"を指定しないと、つぎの順番でアクセスされます。

1　READ WRITE　2　WRITE　3　READ

## ■引数とオプション（書式②）

<table>
<tr><th>引数とオプション</th><th>機　能</th></tr>
<tr><td>モード</td><td>つぎの中からファイルのタイプを指定する。
<table>
<tr><th>モード</th><th>ファイル</th></tr>
<tr><td>"I"</td><td>シーケンシャル・ファイルから読み込む。</td></tr>
<tr><td>"O"</td><td>シーケンシャル・ファイルへ書き出す。<br>ファイルの先頭から書き出す。</td></tr>
<tr><td>"A"</td><td>シーケンシャル・ファイルへ書き出す。<br>ファイルの終わりに追加する。</td></tr>
<tr><td>"R"</td><td>ランダム・アクセス・ファイルをオープンする。</td></tr>
<tr><td>"B"</td><td>バイナリー・ファイルとしてオープンする。</td></tr>
</table></td></tr>
<tr><td>ファイル番号</td><td>オープンしたファイルに割り当てる番号を指定する。<br>オープン済みファイルの指定は、この番号を使う。</td></tr>
<tr><td>ファイル名</td><td>オープンしたいファイルの名前を指定する。</td></tr>
<tr><td>レコード・サイズ</td><td>オープンするファイルのタイプのレコード長を指定する。
<table>
<tr><th>ファイル</th><th>レコード・サイズ</th></tr>
<tr><td>シーケンシャル</td><td>省略可能<br>デフォルトのバッファ・サイズは、<br>512バイトになっている。</td></tr>
<tr><td>ランダム</td><td>１レコードのバイト数を指定する。<br>デフォルトのバッファ・サイズは、<br>128バイトになっている。</td></tr>
<tr><td>バイナリー</td><td>不要</td></tr>
</table></td></tr>
</table>

■文例②

```
TYPE FoodRecType
  itName AS STRING * 10
  amount AS INTEGER
  END TYPE
DIM fruit AS FoodRecType

OPEN "R", 1, "Fruit.Dat", LEN(fruit)
recNum = 1
PRINT " N A M E    amount "
PRINT "--------------------"
DO
  GET #1, recNum, fruit
  IF EOF(1) THEN
    EXIT DO
  END IF
  PRINT fruit.itName; " = "; USING "##,###"; fruit.amount
  recNum = recNum + 1
LOOP
CLOSE 1
END
```

■ノート

▶参照ステートメント：GET #、PUT #、FIELD、CLOSE

# OPEN ”COMn：”[通信]

## 通信用チャンネルのオープン

■機能

通信用のチャンネルをオープンする

■書式

OPEN ”COMn：オプション・リスト1　オプション・リスト2”
　　[FOR モード] AS [#] ファイル番号
　　[LEN = レコード・サイズ]

■引数とオプション

| 引数とオプション | 機　能 |
|---|---|
| n | 通信デバイスの番号(1〜2)を指定する。 |
| オプション・リスト1 | 通信のパラメータについて設定する。<br>[ボーレート][,[パリティー][,[ビット長][,[ストップ・ビット]]]]<br>ボーレート：75〜9600　パリティー：N, E, O　ビット長：5〜8　ストップ・ビット：1, 2 |

| オプション・リスト2 | 制御のためのオプション設定をする。 | |
|---|---|---|
| | オプション | 機　能 |
| | ASC | ASCII モードとしてオープンすることにより、つぎのことを行なえる。<br>※タブをスペースに置き換える<br>※行末に CR(&H0D, &H0A)を付ける<br>※&H1A を EOF として認識する<br>※ XON/XOFF を可能にする |
| | BIN | バイナリー・モードとしてオープンすることにより、つぎのことを行なえる。<br>※タブをスペースに置き換えない<br>※行末に CR を付けない<br>※&H1A を EOF として認識しない |
| | CD [0] | CD を制御する |
| | CS [0] | CTS を制御する |
| | DS [m] | DSR を制御する |
| | LF | 通信内容をプリントする |
| | OP [m] | 待ち時間を設定する。 |
| | RB [n] | 受信バッファのサイズを設定する。 |
| | RS | RTS を制御する。 |
| | ※オプションは、カンマで区切る。<br>※ m の単位は、ミリ・セカンドである。 | |

<table>
<tr><td>モード</td><td colspan="2">オープンの仕方を指定する。</td></tr>
<tr><td></td><td>モード</td><td>機　能</td></tr>
<tr><td></td><td>INPUT</td><td>シーケンシャル・ファイル入力</td></tr>
<tr><td></td><td>OUTPUT</td><td>シーケンシャル・ファイル出力</td></tr>
<tr><td></td><td>RANDOM</td><td>ランダムアクセス入出力</td></tr>
<tr><td>ファイル番号</td><td colspan="2">オープン時のファイル番号を指定する。</td></tr>
<tr><td>レコード・サイズ</td><td colspan="2">ランダムアクセス・バッファ・サイズ を指定する。<br>省略すると128バイトになる。</td></tr>
</table>

## ■文例

```
OPEN "COM1:1200,N,8,1,BIN" AS 1
```

## ■ノート

▶参照ステートメント：COM(n)、ON　event　GOSUB

# OPTION BASE

## 配列変数における添字の下限の設定



■機能

配列変数における添字の下限を設定する

■書式

**OPTION BASE 最小値**

■引数

| 引数 | 機能 |
|---|---|
| 最小値 | 配列の添え字の下限を0または1のどちらかに設定する。 |

■文例

```
OPTION BASE 1
DIM lastDay(12) AS INTEGER
FOR n = 1 TO 12
  READ LastDay(n)
NEXT n
  .
  .
DATA 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31
```

■ノート

▶デフォルトの下限は、0になっている。

# OUT

## 入出力ポートへのデータの送出

### ■機能

入出力ポートに1バイトのデータを送る

### ■書式

**OUT　ポート番号，データ**

### ■引数

| 引数 | 機　　能 |
| --- | --- |
| ポート番号 | 送り出し先のポート番号(0から65535)を指定する。 |
| データ | 送り出すデータ(0～255)を指定する。 |

### ■文例

```
OUT &HAB, &H20
```

### ■ノート

▶参照関数：INP

OUT

# PAINT

## グラフィック領域の塗りつぶしとパターン埋め

### ■機能

任意のグラフィック領域を指定したカラーで塗りつぶすか、またはパターンで埋める

### ■書式

**PAINT　[STEP] (x, y) [, [ペイント属性]**
**[, [境界線のカラー], [, 背景]]]**

### ■引数とオプション

| 引数とオプション | 機　能 |
|---|---|
| STEP | 現在のグラフィック・カーソルの位置から相対的にx、yの座標を決めたいときに指定する。 |
| x, y | 塗りつぶしたい領域に含まれる座標を指定する。 |
| ペイント属性 | 特定のカラーで塗りつぶしたいときはそのカラーの属性番号(数字)で指定する。<br>ある一定のパターンで塗りつぶしたいときは、そのパターンを文字式で指定する。 |
| 境界線のカラー | 境界線のカラーを属性番号で指定する。 |
| 背景 | すでに塗りつぶされている部分を別のパターンで塗りつぶすためのパターンを指定する。 |

## 文例

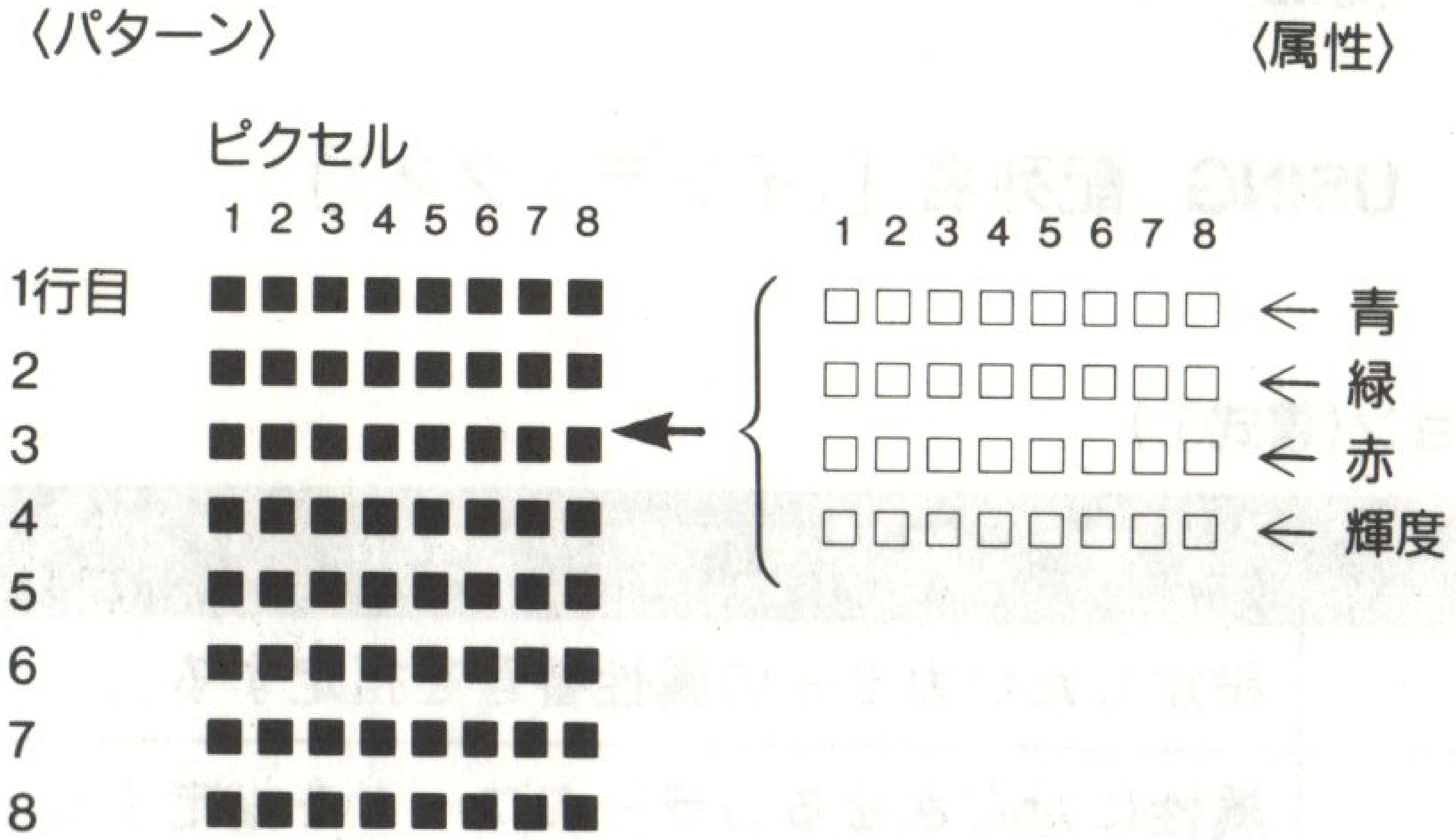
```
row1$ = CHR$(&H0, &HF, &H0, &H0)
row2$ = CHR$(&H0, &HF0, &H0, &H0)
row3$ = CHR$(&H0, &HF0, &H0, &H0)
row4$ = CHR$(&H0, &HF, &H0, &H0)
pixelPattern$ = row1$ + row2$ + row3$ + row4$
LINE (100, 200)-(200, 300), , B
PAINT (150, 250), pixelPattern$
```

## ノート

▶パターンの作り方

※パターンは、タイルを並べた模様になる。

※1つのタイルは8ピクセル×8行になっており、行単位で定義する。




※タイルの1行は、横に8つのピクセルを並べたものである。この1行は4バイトの文字列で定義する。1バイトは8ビットなので、各ビットがピクセルに対応している。

※1行を定義するための4バイトは、つぎの属性になっている。

CHR$(青)＋CHR$(緑)＋CHR$(赤)＋CHR$(輝度)

または

CHR$(青、緑、赤、輝度)

1行は、これら4つの属性を重ね合わせて定義される。

※1パターンの定義は、1行(4バイト)単位で行なえる。1行だけ定義するとそのパターンが残りの行にも適用されるので8行(32バイト)分すべてを定義しなくてもよい。ただし。背景は、2行までである。

▶参照ステートメント：COLOR、LINE、CIRCLE

# PALETTE

## パレット・カラーの設定

■機能

パレットのカラー属性とカラー・コードの関係を設定する。1つずつ設定するときは、書式①を使い、一度に全部を設定したいときは、書式②を使う。

■書式

① PALETTE 属性, カラー

② PALETTE USING 配列名 [(インデックス)]

■引数とオプション(書式①)

| 引数 | 機能 |
|---|---|
| 属性 | 設定したいカラーの属性番号を指定する。 |
| カラー | 属性に対応させるカラーのコードを指定する。 |

■引数とオプション(書式②)

| 引数とオプション | 機能 |
|---|---|
| 配列名 | 属性とカラー・コードとの対応テーブルとして整数型または長整数型の配列を指定する。 |
| インデックス | パレットとして設定するときにどの要素から始めるかを指定する。 |

■文例①

```
PALETTE 1, 6
```

■文例②

```
CONST colorMax = 16
DATA 0, 1, 2, 3, 4, 5, 6, 7, 0, 1, 2, 3, 4, 5, 6, 7
DIM Picasso(1 TO colorMax) AS INTEGER
FOR i = 1 TO colorMax
  READ setColorCode
  Picasso(i) = setColorCode
NEXT i
PALETTE USING Picasso
  .
  .
```

■ノート

▶カラーの属性番号は、テキストのカラーには影響しない。

▶使用可能な属性番号は、接続できるモニターの種類によって異なる。

| タイプ | 前景 | 背景 |
|---|---|---|
| デジタル | 0〜31 | 0〜 7 |
| アナログ | 0〜31 | 0〜15 |

▶カラーの種類は、SCREENステートメントで設定するパレット番号に依存する。

▶デフォルトとして、属性番号は、カラー・コードが設定されている。

▶参照ステートメント：SCREEN、COLOR

# PCOPY

## スクリーン・ページのコピー

### ■機能

あるスクリーン・ページを別のページにコピーする

### ■書式

**PCOPY　コピー元ページ番号，コピー先ページ番号**

### ■引数とオプション

| 引数 | 機　能 |
|---|---|
| コピー元ページ番号 | コピー元のスクリーン・ページ番号を指定する。 |
| コピー先ページ番号 | コピー先のスクリーン・ページ番号を指定する。 |

### ■文例

```
PCOPY 0, 1
```

### ■ノート

▶このステートメントは、アクティブ・ページをビジュアル・ページにコピーしたりするときに使う。

▶参照ステートメント：SCREEN

# POKE

## メモリーへの書き込み

■機能

メモリー上の特定のアドレスに1バイト書き込む

■書式

**POKE　アドレス，バイト**

■引数とオプション

| 引数 | 機　能 |
|---|---|
| アドレス | 書き込み先のオフセット・アドレス(0～65535)を指定する。 |
| バイト | 書き込みたいデータ(0～255)を指定する。 |

■文例

```
FOR addr = &H400 TO &H412
    POKE addr, PEEK(addr + &H100)
NEXT addr
   ・
   ・
   ・
```

■ノート

▶参照ステートメント：DEF　SEG

▶参照関数：PEEK

# PRESET

## ピクセルのカラー設定



■機能

スクリーン上のピクセルにカラーを設定し、描く

■書式

**PRESET ［STEP］(x 座標, y 座標)［, カラー］**

■引数とオプション

| 引数とオプション | 機能 |
|---|---|
| STEP | 現在のグラフィック・カーソルの位置から相対的に x,y の座標を決めたいときに指定する。 |
| x 座標, y 座標 | 設定したいピクセルの座標を指定する。 |
| カラー | カラーの属性を設定する。 |

■文例

```
FOR x = 1 TO 600
  PSET (x, 200), 7
  PRESET STEP(-1, 0)
NEXT x
PRESET (POINT(0) + 1, POINT(1))
```

■ノート

▶カラーを省略すると、そのときのバックグランド・カラーで表示される。

▶参照ステートメント：PSET、COLOR

▶参照関数：POINT

# PRINT [スクリーン]

## スクリーン表示

■機能

スクリーンにデータを表示する

■書式

**PRINT　[USING 書式制御文字列；]　式リスト　[{, | ；}]**

■引数とオプション

| 引数とオプション | 機　能 |
|---|---|
| USING | 書式制御文字列を使うときに指定する。 |
| 書式制御文字列 | 式の値の表示形式や位置を制御したいときに使う。<br>●文字列の表示<br>制御文字 / 機能<br>@ : その位置からすべてを表示する。<br>& : 2つの&の範囲に表示する。<br>! : 最初に1文字のみを表示する。<br>●数値の表示<br>制御文字 / 機能<br># : 右詰めで表示する。<br>, : カンマを表示する。<br>¥ : 先頭に¥を付ける。<br>* : 空白部分を＊で埋める。<br>. : 小数点を表示する。<br>+ : 符号を先頭または末尾に表示する。<br>− : -を先頭に表示する。<br>^ : 指数形式で表示する。<br>_ : 書式制御文字を文字として表示する。 |
| 式リスト | 表示したい値を指定する。<br>式は、",　"または"；"で区切る。 |

■文例①

```
PRINT "Quick","BASIC"
PRINT "Quick";"BASIC"
PRINT
PRINT "Quick";
PRINT "BASIC"
```

〈表示結果〉

```
Quick         BASIC
QuickBASIC

QuickBASIC
```

■文例②

```
PRINT USING "@BASIC"; "Quick"
PRINT USING "& &BASIC"; "Quick"
PRINT USING "!BASIC"; "Quick"
PRINT USING "########## 円"; 4200
PRINT USING "##,###,### 円"; 4200
PRINT USING "¥¥,###,###"; 4200
PRINT USING "**,###,###"; 4200
PRINT USING "QucikBASIC(#.##); 4.2
PRINT USING "+#.##"; -4.2
PRINT USING "#.##-"; -4.2
PRINT USING "#.#^^^^"; 4.2
PRINT USING "#######,#"; 420000000
PRINT USING "#.## "; 3.1; 5.7; .38
```

〈表示結果〉

```
QuickBASIC
QuiBASIC
QBASIC
      4200 円
     4,200 円
    ¥4,200
*****4,200
QucikBASIC(4.20)
-4.20
4.20-
0.4E+01
%420,000,000
3.10 5.70 0.38
```

■ノート

▶引数およびオプションを省略すると、改行する。

▶ステートメントの最後に","または";"を付けると改行しないで次の表示を行なう(USINGでは";"だけ使える)。

▶USINGを使わない式リスト中での式は、等間隔な表示を行なうには","で区切り、また続けて表示するには";"で区切らなければならない。

▶USINGを使った式リスト中での式は、";"で区切らなければならない。

▶","は、"#"の中に1つあれば3桁ごとに区切ることができる。

▶オバーフローすると先頭に"%"が表示される。

▶1つの書式制御文字列で、2つ以上の式を同じ書式で表示できる。

▶書式制御文字列には、文字列変数が使える。

# PRINT # [ファイル]

## シーケンシャル・ファイルへの書き出し

### ■機能

シーケンシャル・ファイルにデータを書き出す

### ■書式

**PRINT #ファイル番号,**
**[USING 書式制御文字列;] 式リスト [{, | ;}]**

### ■引数とオプション

| 引数とオプション | 機能 |
|---|---|
| #ファイル番号 | OPEN ステートメントでオープンしたときのファイル番号を指定する。 |
| USING | 書式制御文字列を使うときに指定する。 |
| 書式制御文字列 | 式の値の表示形式や位置を制御したいときに使う。<br>●文字列の表示 |

| 制御文字 | 機能 |
|---|---|
| @ | その位置からすべてを表示する。 |
| & | 2つの&の範囲に表示する。 |
| ! | 最初に1文字のみを表示する。 |

●数値の表示

| 制御文字 | 機　　能 |
|---|---|
| # | 右詰めで表示する。 |
| , | カンマを表示する。 |
| ¥¥ | 先頭に¥を付ける。 |
| ＊＊ | 空白部分を＊で埋める。 |
| . | 小数点を表示する。 |
| + | 符号を先頭または末尾に表示する。 |
| - | –を先頭に表示する。 |
| ^ | 指数形式で表示する。 |
| _ | 書式制御文字を文字として表示する。 |

| | |
|---|---|
| 式リスト | 表示したい値を指定する。<br>式は、",　"または"；"で区切る。 |

■ノート

▶文例とノートについては、PRINTステートメントの箇所を参照。出力先がスクリーンではなく、ディスク上のファイルになる。

# PSET

## ピクセルのカラー設定

■機能

スクリーン上のピクセルにカラーを設定し描く

■書式

**PSET [STEP] (x 座標, y 座標) [, カラー]**

■引数とオプション

| 引数とオプション | 機　能 |
|---|---|
| STEP | 現在のグラフィック・カーソルの位置から相対的に x、y の座標を決めたいときに指定する。 |
| x 座標, y 座標 | 設定したいピクセルの座標を指定する。 |
| カラー | カラーの属性を設定する。 |

■文例

```
FOR x = 1 TO 600
  PSET (x, 200), 7
  PRESET STEP(-1, 0)
NEXT x
PRESET (POINT(0) + 1, POINT(1))
```

■ノート

▶カラーを省略すると、そのときのフォアグランド・カラーで表示される。

▶参照ステートメント：PRESET、COLOR

▶参照関数：POINT

# PUT [ファイル]

## ランダムアクセス・ファイル/バイナリー・ファイルへの書き出し



■機能

ランダム・アクセス・ファイルまたはバイナリー・ファイルにデータを書き出す

■書式

**PUT [#] ファイル番号 [, [レコード番号] [, 変数]]**

■引数とオプション

| 引数とオプション | 機　能 |
|---|---|
| ファイル番号 | OPEN ステートメントでオープンしたときのファイル番号を指定する。 |
| レコード番号 | ランダムアクセス・ファイルのときは、次に書き出すレコードの番号(1〜2147483647)を指定する。<br>バイナリー・ファイルのときは、次に書き出すバイトの開始位置(1〜ファイル・サイズ)を指定する。 |
| 変数 | 書き出すデータを格納するための変数を指定する。 |

■文例①

```
TYPE FoodRecType
  itName AS STRING * 10
  amount AS INTEGER
  END TYPE
DIM fruit AS FoodRecType

OPEN "Fruit.Dat" FOR RANDOM AS 1 LEN = LEN(fruit)
recNum = 1
```

```
DO
  INPUT ; " Fruit (<cr> for EXIT) = ", fruit.itName
  IF fruit.itName = SPACE$(10) THEN
    EXIT DO
  END IF
  INPUT " , amount = ", fruit.amount
  PUT #1, recNum, fruit
  recNum = recNum + 1
LOOP
CLOSE 1
END
```

■文例②

```
' ファイルのコピー
DIM oneByte AS STRING * 1
' enter file names ...
INPUT "Source file name = ", sourceFile$
INPUT "Destination file name = ", destFile$
' open files ...
OPEN sourceFile$ FOR BINARY AS 1
OPEN destFile$ FOR BINARY AS 2
bytePos = 1
DO
  GET #1, bytePos, oneByte
  IF EOF(1) THEN
    EXIT DO
  END IF
  PUT #2, bytePos, oneByte
  bytePos = bytePos + 1
LOOP
CLOSE 1, 2
END
```

■ノート

▶"変数"にレコード型変数を使うときは、その宣言名だけでよい。

▶FIELDステートメントによってバッファを割り当てるときは、"変数"の部分は省く(FIELDステートメントを参照)。

▶バイナリー・ファイルへは、その変数のサイズに等しいバイト数だけ書き出される。

▶参照ステートメント：OPEN、GET #、FIELD、CLOSE

# PUT［グラフィックス］

## スクリーン・イメージの描画

■機能

GET ステートメントで読み込んだスクリーン・イメージをスクリーンに描く

■書式

PUT ［STEP］ (x, y), 配列変数名 ［(配列要素)］
［, ピクセル操作］

■引数とオプション

| 引数とオプション | 機能 |
|---|---|
| STEP | 現在のグラフィック・カーソルの位置から相対的に x、y の座標を決めたいときに指定する。 |
| x, y | そのイメージの四角形の左上座標を指定する。 |
| 配列変数名 | GET ステートメントで保存した配列の名前を指定する。 |
| 配列要素 | 配列上のどの要素から使うかを指定する。 |
| ピクセル操作 | すでにグラフィック・スクリーンに描かれている図との関係を指定する。 |

| 操作 | 用途 |
|---|---|
| PSET | そのまま表示する。 |
| PRESET | 反転して表示する。 |
| AND | AND 操作を行なう。 |
| OR | OR 操作を行なう。 |
| XOR | XOR 操作を行なう。 |

■文例

```
READ x1, y1, x2, y2
DATA 1, 1, 100, 100
bytes = 4 + 4 * INT(((x2 - x1 + 1) + 7) / 8) * (y2 - y1 + 1)
DIM imageSample(bytes / 2) AS INTEGER
SCREEN 88, , 0, 1
LINE (x1, y1)-(x2, y2), , BF
GET (x1, y1)-(x2, y2), imageSample
FOR n = 1 TO 10
  PUT (n * 10, n * 10), imageSample, XOR
  PCOPY 0, 1
NEXT n
  .
  .
  .
```

■ノート

▶"ピクセル操作"を省略すると、XOR を指定したのと同じになる。

▶参照ステートメント：GET、BSAVE、BLOAD

# RANDOMIZE

## 乱数発生の初期化

■機能

乱数の発生を初期化する

■書式

**RANDOMIZE ［式］**

■オプション

| オプション | 機　能 |
|---|---|
| 式 | 乱数発生のもとになる系列(seed)を-32768〜32767の範囲内で指定します。 |

■文例①

```
RANDOMIZE 5
FOR i = 1 to 10
  PRINT INT(RND * 2)
NEXT i
```

■文例②

```
新しい乱数系列を入力してください (-32768 to 32767)->?
```

■ノート

▶”式”を省略するとその入力プロンプトが表示されます(文例②)。

▶ RND関数を使うプログラムでは、実行の最初で初期化しないと同じ乱数系列になる。

▶参照関数：RND

# READ

## DATAステートメントのデータの読み込み

■機能

DATAステートメントで定義されているデータを読み込んで変数に代入する。

■書式

**READ 変数リスト**

■引数

変数リスト

| 引数 | 機　能 |
|---|---|
| 変数リスト | 読み込むための変数を指定する。 |

■文例

```
DIM dayName(6) AS STRING * 9
FOR n = 0 TO 6
  READ dayName(n)
NEXT n
  .
  .
  .
DATA Sunday, Monday, Tuesday, Wednesday
DATA Thursday, Friday, Saturday
  .
  .
```

■ノート

▶参照ステートメント：DATA、RESTORE

# REDIM

## 配列空間の変更

■機能

$DYNAMIC 宣言した配列のアロケーション空間を再度割り当てる。

■書式

REDIM　[SHARED]　変数(添字)　[AS　タイプ]
　　　[, 変数(添字)　[AS　タイプ]]　...

■引数とオプション

| 引数とオプション | 機　能 |
|---|---|
| SHARED | モジュール・レベルの変数をそのモジュールに含まれるすべてのプロシージャとの間で共有化したいときに指定する。 |
| 変数(添字) | 宣言したい配列変数名とその次元および範囲を指定する。添字は、以下のようにして最大60次元まで指定できる。<br>(添字 [, 添字, ...])<br>1つの添字の範囲(-32768〜32767)は、<br>[下限　TO]　上限<br>の形式で指定する。 |

| | |
|---|---|
| タイプ | 変数のタイプを指定する。このタイプ指定を行なうと変数名には、型宣言文字(%、&、!、#、$)を付ける必要はない。 |

| タイプ名 | 機　能 |
|---|---|
| INTEGER | 整数型 |
| LONG | 長整数型 |
| SINGLE | 単精度型 |
| DOUBLE | 倍精度型 |
| STRING | 固定長文字列(n は文字長)<br>STRING　*　n |
| ユーザー定義タイプ | ユーザー定義タイプ |

■文例

```
'$DINAMIC
REDIM temporaryArea(1 TO 1000) AS INTEGER
```

■ノート

▶添字の大きさは変更できるが、その次元については変更できない。

▶参照メタ・コマンド：$DYNAMIC

▶参照ステートメント

SHARED、COMMON、STATIC、OPTION　BASE

▶参照関数：LBOUND、UBOUND

# REM

## 注釈

■機能

注釈を書く

■書式

REM ［注釈］

■オプション

| オプション | 機　能 |
|---|---|
| 注釈 | 注釈文(コメント)を書く。 |

■文例

```
REM メール管理プログラム
REM ---------------------
DECLARE SUB SendLetter (who$)      ' プロシージャ
DECLARE SUB RecieveLetter (who$)  ' プロシージャ
    .
    .
    .
    .
```

■ノート

▶ REMの代わりに、' が使える。

▶ ' は、ステートメントの終わりに、":"で区切らずに注釈を書くことができる。

# RESET

## ファイルのクローズ

**■機能**

現在オープンしているすべてのディスク・ファイルをクローズする

**■書式**

**RESET**

**■引数とオプション**

なし

**■文例**

```
RESET
```

**■ノート**

▶参照ステートメント：CLOSE

# RESTORE

## 読み込むべきDATAステートメント行の指定

■機能

READステートメントで読み込むDATAステートメント行を指定する。このステートメントを使うと、同じREADステートメントのデータを何回でも読み込むことができる。

■書式

**RESTORE ［{行ラベル｜行番号}］**

■オプション

| オプション | 機　能 |
|---|---|
| 行ラベル<br>行番号 | 次に実行するREADステートメントで読み込むDATAステートメントの行を指定する。 |

■文例

```
DIM lastDay(12) AS INTEGER
RESTORE EndOfMonth
FOR n = 1 TO 12
  READ LastDay(n)
NEXT n
  .
  .
DayNameData:
  DATA Sunday, Monday, Tuesday, Wednesday
  DATA Thursday, Friday, Saturday
EndOfMonth:
  DATA 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31
```

■ノート

▶オプションを指定しないと、最初のDATAステートメントのデータから読み込まれる。

▶参照ステートメント：READ、DATA

# RESUME

## プログラムの続行

■機能

エラー処理したあと、プログラムの実行を続行する

■書式

**RESUME　{[0]　|NEXT|　{行番号|行ラベル}}**

■引数とオプション

| 引数とオプション | 機　能 |
|---|---|
| 0 | 0を指定するか省略するとエラー発生ステートメントから実行を開始する。 |
| NEXT | これを指定すると、エラー発生ステートメントの次ステートメントから実行を開始する。 |
| 行番号<br>行ラベル | 指定した行から開始する。 |

■文例

```
RESUME NEXT
ErrorHandleRoutine:
    .
    .
    .
  RESUME NEXT
```

■ノート

▶"行番号"と"行ラベル"は、モジュール・レベルでなければならない。

▶参照ステートメント：ERROR、ON　ERROR

▶参照関数：ERR、ERL

# RETURN

## サブ・ルーチンの終了



■機能

サブ・ルーチンから戻る

■書式

**RETURN　[{行番号|行ラベル}]**

■オプション

| オプション | 機　能 |
|---|---|
| 行番号 | 戻り先の行番号または行ラベルを指定する。<br>行番号は、0から65529の範囲でなけれならない。 |
| 行ラベル | 行ラベルについてのルール<br>・文字で始めなければならない<br>・":"で終わらなければならない<br>・40文字以内でなければならない<br>・大文字と小文字の区別はない |

■文例

```
GetNameRoutine:
  INPUT "Your Name ?" , theName$
  RETURN
```

■ノート

▶行番号および行ラベル省略をすると、呼び出したGOSUBのつぎのステートメントに戻る。

▶戻るときに、必ずRETURNステートメントを使って戻らなければならず、GOTOステートメントで戻ってはいけない。

# RMDIR

## ディレクトリの削除

■**機能**

ディレクトリを削除する

■**書式**

**RMDIR　パス名**

■**引数**

| 引数 | 機　能 |
|---|---|
| **パス名** | **削除したいパス名を文字式で指定する。** |

■**文例**

```
RMDIR "MAILING"
thePath$ = "B:¥MODULES¥DOSCALL"
RMDIR thePath$
```

■**ノート**

▶ディレクトリの作成は、MKDIR ステートメントを使う。

▶ディレクトリの変更は、CHDIR ステートメントを使う。

# RSET

## 文字列変数への右詰め代入

■機能

文字列変数に右詰めで代入する

■書式

**RSET 文字列変数 = 文字式**

■引数

| 引数 | 機　能 |
|---|---|
| 文字列変数 | 文字列変数やランダムアクセス・バッファの変数を指定する。 |
| 文字式 | 右詰めにしたい値の文字式を指定する。 |

■文例

```
OPEN "Fruit.Dat" FOR RANDOM AS 1 LEN = 12
FIELD #1, 10 AS nameFld$, 2 AS salesFld$
recNum = 1
DO
  INPUT ; " Fruit (<cr> for EXIT) = ", itName$
  IF itName$ = "" THEN
    EXIT DO
  END IF
  INPUT " , amount = ", amount%
  LSET nameFld$ = itName$
  RSET salesFld$ = MKI$(amount%)
  PUT #1, recNum
  recNum = recNum + 1
LOOP
CLOSE 1
END
```

■ノート

▶ FIELDバッファに数値を書き込むには、MKI$、MKL$、MKS$、MKD$関数を使って文字列にしてから行なう。

# RUN

## プログラムの実行

**■機能**

メモリー上またはディスク上のプログラムを実行する

**■書式**

**RUN　[{行番号｜プログラム名}]**

オプション

| オプション | 機　能 |
|---|---|
| 行番号 | 行番号を指定すると、メモリーの常駐しているQuick BASIC のプログラムをその行番号から実行する。省略すると先頭から実行する。 |
| プログラム名 | 実行したいプログラム名を指定する。拡張子を省略して、Quick　BASIC の環境内でこのステートメントを実行すると".BAS"とみなし、また実行可能形式のプログラムの中では、".EXE"とみなす。 |

**■文例**

```
RUN "WordPlus"
```

**■ノート**

▶このステートメントは、実行前にオープン中のデータ・ファイルを自動的にクローズし、かつメモリー上のデータをクリアする。

▶参照ステートメント：CHAIN

# SCREEN

## スクリーン・モードの設定

### ■機能

スクリーンのモードを設定する

### ■書式

**SCREEN　［モード番号］，［パレット番号］，**
**［アクティブ・ページ］，［ビジュアル・ページ］**

### ■オプション

<table>
<tr><th>オプション</th><th colspan="2">機　能</th></tr>
<tr><td rowspan="6">モード番号</td><td colspan="2">スクリーンのモードを番号で指定する。</td></tr>
<tr><th>番号</th><th>種類　(　)内はグラフィック解像度を示す</th></tr>
<tr><td>81</td><td>テキスト</td></tr>
<tr><td>84</td><td>テキストとグラフィック(640×200)</td></tr>
<tr><td>87</td><td>グラフィック(640×400)</td></tr>
<tr><td>88<br>0</td><td>テキストとグラフィック(640×400)</td></tr>
</table>

| | |
|---|---|
| パレット番号 | パレットで使える色数を番号で指定する。<br>番号 / 色数<br>1 / 16 色<br>2 / 64 色<br>3 / 4096 色 |
| アクティブ・ページ | グラフィック関係のステートメントの対象になるグラフィック・スクリーンのページ番号を指定する。 |
| ビジュアル・ページ | 実際に見えるグラフィック・スクリーンのページ番号を指定する。 |

| 番号 | 色数 |
|---|---|
| 1 | 16 色 |
| 2 | 64 色 |
| 3 | 4096 色 |

SCREEN

■文例

```
SCREEN 88, , 1, 0
```

■ノート

▶モードを指定すると、スクリーンはクリアされる。

▶モード87では、テキストはグラフィック・スクリーンに描かれる。

▶参照ステートメント：COLOR、PALETTE

# SEEK

## ファイル上の位置の設定

■機能

オープンしているファイルに対してデータの読み書き(入出力)を行なうために、そのファイル上の特定の位置に設定する

■書式

**SEEK [#]ファイル番号 , 位置**

■引数とオプション

| 引数 | 機　能 |
|---|---|
| ファイル番号 | OPEN ステートメントでオープンしたときのファイル番号を指定する。 |
| 位置 | ファイル上の位置を指定する。<br>ランダムアクセス・ファイルではレコード番号で、その他のファイルでは先頭からのバイト数で指定する |

## ■文例

```
TYPE FoodRecType
  itName AS STRING * 10
  amount AS INTEGER
  END TYPE
DIM fruit AS FoodRecType

OPEN "Fruit.Dat" FOR RANDOM AS 1 LEN = LEN(fruit)
recMax = LOF(1) / LEN(fruit)
enterPrompt$ = "Rec. No (1 -" + STR$(recMax) + ") "
DO
  ' enter RecNo ...
  PRINT enterPrompt$;
  INPUT recNum
  ' branching ...
  SELECT CASE recNum
  CASE 0
      EXIT DO
  CASE IS > recMax
      PRINT "Not valid record No. !"
  CASE ELSE
      SEEK #1, recNum
      GET #1, , fruit
      PRINT fruit.itName; " = "; USING "##,###"; fruit.amount
  END SELECT
LOOP
```

## ■ノート

▶参照ステートメント：OPEN、GET #

▶参照関数：SEEK

SEEK

# SELECT CASE

## 多重分岐

### ■機能

多重分岐のブロックのうちの1つを実行する

### ■書式

```
SELECT CASE テスト式
    CASE 式リスト1
        [ステートメント・ブロック1]
    [CASE 式リスト2
        [ステートメント・ブロック2]]
    ⋮
    [CASE ELSE
        [ステートメント・ブロックn]]
END SELECT
```

### ■引数とオプション

| 引数とオプション | 機能 | |
|---|---|---|
| テスト式 | テストしたい値をもつ式を書く。 | |
| 式リスト1～m | テスト式が取りうる値、範囲、または関係式を書く。 | |
| | 指定の仕方 | 例 |
| | 等しい値 | CASE 1, 3, 5, 7, 8, 10, 12 |
| | 特定の範囲値 | CASE 1 TO 5, "FMT" TO "HIT"<br>(TOを使う) |

| | | |
|---|---|---|
| | 関係式 | CASE IS = 5<br>CASE IS <> 5<br>CASE IS < 5<br>CASE IS <= 5<br>CASE IS > 5<br>CASE IS >= 5<br>(IS を使う) |
| | 1つの式リストにこれらを混在できる。 | |
| ステートメント・ブロック1〜m | 式リスト1〜mが真のときに実行させたいステートメントを書く。 | |
| ステートメント・ブロックn | 式リスト1〜mのすべてが偽のときに実行させたいステートメントを書く。 | |

## ■文例

```
SUB binarySearch (isStat, theArray(), keyData)
  low = LBOUND(theArray)
  high = UBOUND(theArray)
  DO WHILE (low <= high)
    mid = INT((low + high) / 2)
    SELECT CASE keyData
    CASE theArray(mid)          ' Found !
      isStat = mid
      EXIT DO
    CASE IS < theArray(mid)     ' search backward
      high = mid - 1
    CASE IS > theArray(mid)     ' search forward
      low = mid + 1
    CASE ELSE                   ' Not found !
      isStat = -1
      EXIT DO
    END SELECT
  LOOP
END SUB
```

## ■ノート

▶"CASE　式2"以下の部分は省略できる。

▶"END　SELECT"の部分は省略できない。

▶式リストに書いたもののうち、どれかが真であればそのステートメント・ブロックを実行する。

▶参照ステートメント：IF

# SHARED

## 変数の共有化

### ■機能

SUBまたはFUNCTIONプロシージャ内において、モジュール・レベルの変数を共有のものとして宣言する。

これにより、モジュール・レベルとそのプロシージャとの間でのみ共有化が可能となる。他のプロシージャがその変数を共有するかどうかは関与しない。

### ■書式

**SHARED 変数 [AS タイプ][, 変数 [AS タイプ]] ...**

### ■引数とオプション

<table>
<tr><th>引数とオプション</th><th colspan="2">機　能</th></tr>
<tr><td>変数</td><td colspan="2">共有化したい変数の名前を指定する。</td></tr>
<tr><td rowspan="8">タイプ</td><td colspan="2">共有化したい変数のタイプを指定する。</td></tr>
<tr><th>タイプ名</th><th>機　能</th></tr>
<tr><td>INTEGER</td><td>整数型</td></tr>
<tr><td>LONG</td><td>長整数型</td></tr>
<tr><td>SINGLE</td><td>単精度型</td></tr>
<tr><td>DOUBLE</td><td>倍精度型</td></tr>
<tr><td>STRING</td><td>固定長文字列(nは文字長)<br>STRING ＊ n</td></tr>
<tr><td>ユーザー定義タイプ</td><td>ユーザー定義タイプ</td></tr>
</table>

■文例①

```
SystemOldIndex = 0
    .
    .
CALL PointRecOne (theKey$)
    .
    .
    .
```

■文例②

```
SUB PointRecOne (itKey$)
    SHARED SystemOldIndex
      .
    SystemOldIndex = recNo
      .
END SUB
```

■ノート

▶参照ステートメント：COMMON、STATIC、DIM

# SHELL

## DOSコマンドの実行



### ■機能

DOS コマンドを実行する

### ■書式

**SHELL ［コマンド文字列］**

### ■引数とオプション

| オプション | 機　能 |
|---|---|
| コマンド文字列 | 実行したいコマンドおよびパラメーターを文字列にして指定する。 |

### ■文例

```
SHELL "format b:/v"
```

### ■ノート

▶オプションを省略すると、"COMMAND.COM"を呼び出す。Quick BASIC に戻るには、DOS プロンプトにおいて、EXIT コマンドを実行する。

▶実行するコマンドは、DOS 内部コマンド、.COM、.EXE、.BAT の順に探す。

▶参照ステートメント

CHDIR、MKDIR、RMDIR、NAME、ENVIRON

# STATIC

## ローカル変数の宣言

### ■機能

SUBおよびFUNCTIONプロシージャ、ならびにDEF　FN関数内で、モジュール・レベルのグローバル変数と同名のローカル変数を宣言する。これにより、不用意なデータ変更を避けることが可能になる

### ■書式

**STATIC　変数［()］［AS　タイプ］［,　変数［()］［AS　タイプ］］...**

### ■引数

| 引数とオプション | 機　能 | |
|---|---|---|
| 変数 | ローカル変数として宣言したい変数を指定する。 | |
| タイプ | 変数のタイプを指定する。このタイプ指定を行なうと変数名には、型宣言文字(%、&、!、#、$)を付ける必要はない。 | |
| | **タイプ名** | **機　能** |
| | INTEGER | 整数型 |
| | LONG | 長整数型 |
| | SINGLE | 単精度型 |
| | DOUBLE | 倍精度型 |
| | STRING | 固定長文字列(nは文字長)<br>STRING　*　n |
| | ユーザー定義タイプ | ユーザー定義タイプ |

STATIC

■文例①

```
SHARED theName AS STRING * 20, SystemParam AS INTEGER
theName = "QuickBASIC"
```

■文例②

```
SUB GetUserName (hisName AS STRING * 20)
  STATIC theName AS STRING * 20
  theName = hisName
     .
     .
     .
END SUB
```

■ノート

▶参照ステートメント：DIM、COMMOM、SHARED

# STOP

## プログラムの終了

### ■機能

Quick BASICの統合環境下で実行しているときは、そのプログラムを終了する。また、実行可能ファイル形式(.EXE)のプログラムを実行しているときは、そのプログラムを終了しDOSへ制御を戻す

### ■書式

**STOP**

### ■引数とオプション

なし

### ■文例

```
STOP
```

### ■ノート

▶参照ステートメント：END、SYSTEM

STOP

# SUB

## SUBプロシージャの定義

■機能

SUBプロシージャを定義する。1つのSUBプロシージャは、SUBステートメントと対応するEND SUBステートメントまでである

■書式

```
SUB プロシージャ名 [(引数リスト)] [STATIC]
      :
END SUB
```

■引数とオプション

| 引数とオプション | 機能 |
|---|---|
| プロシージャ名 | SUBプロシージャ名を40文字以内で指定する。 |
| 引数リスト | SUBプロシージャがコール側からのパラメーターを受け取るための引数リストである。<br>この引数は、<br>変数 [()] [AS タイプ]<br>↑ SUB内で使う変数名<br>↑ 配列用<br>↑ つぎのどれかで指定する<br>・INTEGER(整数)<br>・LONG (長整数)<br>・SINGLE(単精度)<br>・DOUBLE(倍精度)<br>・STRING(可変長文字列)<br>・ユーザー定義タイプ<br>からなり、複数あるときはカンマで区切る。 |
| STATIC | これを指定すると、そのローカル変数は保たれる。 |

■文例（配列データを使ったバイナリー・サーチ）

```
SUB binarySearch (isStat, theArray(), keyData)
  low = LBOUND(theArray)
  high = UBOUND(theArray)
  WHILE (low <= high)
    mid = INT((low + high) / 2)
    IF (keyData < theArray(mid)) THEN
      high = mid - 1
    ELSEIF (keyData > theArray(mid)) THEN
      low = mid + 1
    ELSE
      isStat = mid
      EXIT SUB
    END IF
  WEND
  isStat = -1
END SUB
```

SUB

■ノート

▶コール側の引数の数およびタイプと同じでなければならない。

▶引数リストにおける変数の名前は、コール側の変数名と同じである必要はない。

▶引数は、参照渡し(call by reference)されるので、プロシージャ内での書換えが可能である。

▶引数リストにおける変数は、SUB内においてローカルであるのでコール側に戻るとき自動的に消滅する。

▶引数リストの中で配列変数を使うときは、その変数名に続けて()のみを付ける。

▶ END SUB以前でコール側に戻るには、EXIT SUBステートメントを使う。

▶呼び出すには、CALLステートメントを使う。

# SWAP

## 変数の値の入れ替え

■機能

2つの変数の値を相互に入れ替える

■書式

**SWAP 変数1, 変数2**

■引数

| 引数 | 機　能 |
|---|---|
| 変数1<br>変数2 | 入れ替えたい2つの変数を指定する。<br>これらの変数は、同じタイプでなければならない。 |

■文例①

```
SWAP firstName$ , secondName$
```

■文例②

```
TYPE aPersonalRecordType              ' ユーザー定義型
  firstName AS STRING * 10
  secondName AS STRING * 10
END TYPE
DIM player AS aPersonalRecordType, pianist AS aPersonalRecordType

LET player.firstName = "Bill"
LET player.secondName = "Evans"
LET pianist.firstName = "David"
LET pianist.secondName = "Matthews"
SWAP player , pianist
```

■ノート

▶ユーザー定義型の変数の入れ替えは、両方のタイプが同じならば、それらを構成している要素を個別に入れ替えなくても変数名だけで行なえる(文例②)。

# SYSTEM

## プログラムの終了

### ■機能

Quick BASICの統合環境下で実行しているときは、そのプログラムを終了する。また、実行可能ファイル形式(.EXE)のプログラムを実行しているときは、そのプログラムを終了しDOSへ制御を戻す

### ■書式

**SYSTEM**

### ■引数とオプション

なし

### ■文例

```
SYSTEM
```

### ■ノート

▶オープン中のファイルは、終了する前にすべてクローズされる。

▶ダイレクト・モード・ウィンドウにおけるこのステートメントの実行は、"F/ファイル"メニューにおける"X/終了"コマンドの実行と同じである。

SYSTEM

# TIME$

## 時刻の設定



### ■機能

MS-DOSの時刻を設定する

### ■書式

**TIME$ = 時刻**

### ■引数

| 引数 | 機能 |
|---|---|
| 時刻 | 設定したい時刻を文字式で指定する。<br>時刻は、"HH:MM:SS"<br>時(00〜23)→HH<br>分(00〜59)→MM<br>秒(00〜59)→SS<br>の形式で指定する。<br>なお、":MM:SS"または":SS"の部分は省略可能であり、その部分には0分0秒または0秒が設定される。 |

### ■文例

```
TIME$ = "15"                  ' 15:00:00 と同じ
TIME$ = "15:07"               ' 15:07:00 と同じ
TIME$ = "15:07:30"
```

### ■ノート

- MS-DOSのTIMEコマンドと同じ機能をもっている。
- MS-DOS上の現在の時刻を知るには、TIME$関数を使う。

# TIMER

## タイマー・イベント・トラッピングの設定

■機能

タイマー・イベント・トラッピングを有効または無効にする

■書式

**TIMER　{ON | OFF | STOP}**

■引数

| 引数 | 機　　能 |
|---|---|
| ON | タイマー・イベント・トラッピングを有効にする。<br>有効であるあいだ、ON TIMER(n) GOSUB ステートメントで設定した秒数が経過したかどうかチェックし、経過したときは、そのイベント・トラッピングのサブルーチンを実行する。 |
| OFF<br>STOP | タイマー・イベント・トラッピングを無効にする。<br>OFF は、そのイベントを記憶しないが、STOP は、記憶しているのでトラッピングできるようになると ON TIMER ルーチンを実行する。 |

■文例

```
TIMER ON
ON TIMER(5) GOSUB skipToNext
DO
LOOP

skipToNext:
TIMER OFF
```

■ノート

▶参照ステートメント：ON　event　GOSUB

# TRON/TROFF

## トレースの開始と終了

■機能

プログラム実行のトレースを開始する。またはプログラム実行のトレースを終える

■書式

① **TRON**

② **TROFF**

■引数とオプション

なし

■文例

```
TRON
```

```
TROFF
```

■ノート

▶トレース中は、実行しているステートメントが反転表示される。

# TYPE

## ユーザー定義型変数の型の定義

■機能

ユーザー定義型変数(レコード変数)の型を定義する。これにより、複雑に構造化したデータ構造を定義できる

■書式

```
TYPE  ユーザー定義タイプ名
          要素名1  AS  タイプ
          要素名2  AS  タイプ
        ⋮
END  TYPE
```

■引数

| 引数 | 機能 |
|---|---|
| ユーザー定義タイプ | 定義したいレコード変数の名前を指定する。 |
| 要素名n | 各要素の名前を指定する。 |
| タイプ | 各要素のタイプをタイプ名で指定する。 |

| タイプ名 | 機能 |
|---|---|
| INTEGER | 整数型 |
| LONG | 長整数型 |
| SINGLE | 単精度型 |

TYPE

| | | |
|---|---|---|
| | DOUBLE | 倍精度型 |
| | STRING | 固定長文字列(n は文字長)<br>STRING ＊ n |
| | ユーザー定義タイプ | ユーザー定義タイプ |

TYPE

## ■文例

```
TYPE RecPointerType                          ' ポインター・タイプ
  ptr AS INTEGER
END TYPE
TYPE LogicalType                             ' ロジカル・タイプ
  YesNo AS INTEGER
END TYPE
TYPE DateType                                ' 日付タイプ
  Year AS INTEGER
  month AS INTEGER
  day AS INTEGER
END TYPE
TYPE MailType                                ' メール・タイプ
  who AS STRING * 20
  where AS STRING * 40
  when AS DateType
  what AS STRING * 30
  reply AS LogicalType
  nextNode AS RecPointerType
ENDTYPE
CONST DONE = -1, UNDO = NOT DONE
DIM sendMail as MailType , recieveMail as MailType
  .
  .
LET sendMail.who = "Chizuko OOHATA"
LET sendMail.when.Year = 1989
LET sendMail.reply.YesNo = DONE
  .
  .
```

## ■ノート

▶参照ステートメント

DIM、REDIM、COMMON、STATIC、SHARED、LET

# UNLOCK

## ファイル・アクセスのアンロック

### ■機能

ネットワークのもとでオープンされているファイルに対するアクセスをアンロックする

### ■書式

**UNLOCK　[#] ファイル番号**
**[, {レコード | [開始位置]　TO　終了位置}]**

### ■引数とオプション

| 引数とオプション | 機　　能 |
|---|---|
| ファイル番号 | アンロックしたいファイルの番号を指定する。 |
| レコード | アンロックしたいレコードまたはバイト位置を指定する。 |
| 開始位置<br>終了位置 | アンロックしたい範囲をレコードまたはバイト位置で指定する。 |

### ■文例

```
UNLOCK #1, 1 TO 100
UNLOCK #2
```

UNLOCK

■ノート

▶ランダムアクセス・ファイルでは、レコード番号で行なう。

▶バイナリー・ファイルでは、バイト位置で行なう。

▶シーケンシャル・ファイルでは、ファイル全体がアンロックされる。

▶ファイル番号だけを指定すると、ファイル全体をアンロックする。

▶"TO 終了位置"だけを指定すると、ファイルの先頭からそこまでをアンロックする。

▶アンロックするときの範囲指定は、ロック時と同じでなければならない

▶参照ステートメント：LOCK

# VIEW

## グラフィックの物理ビュー・ポートの定義



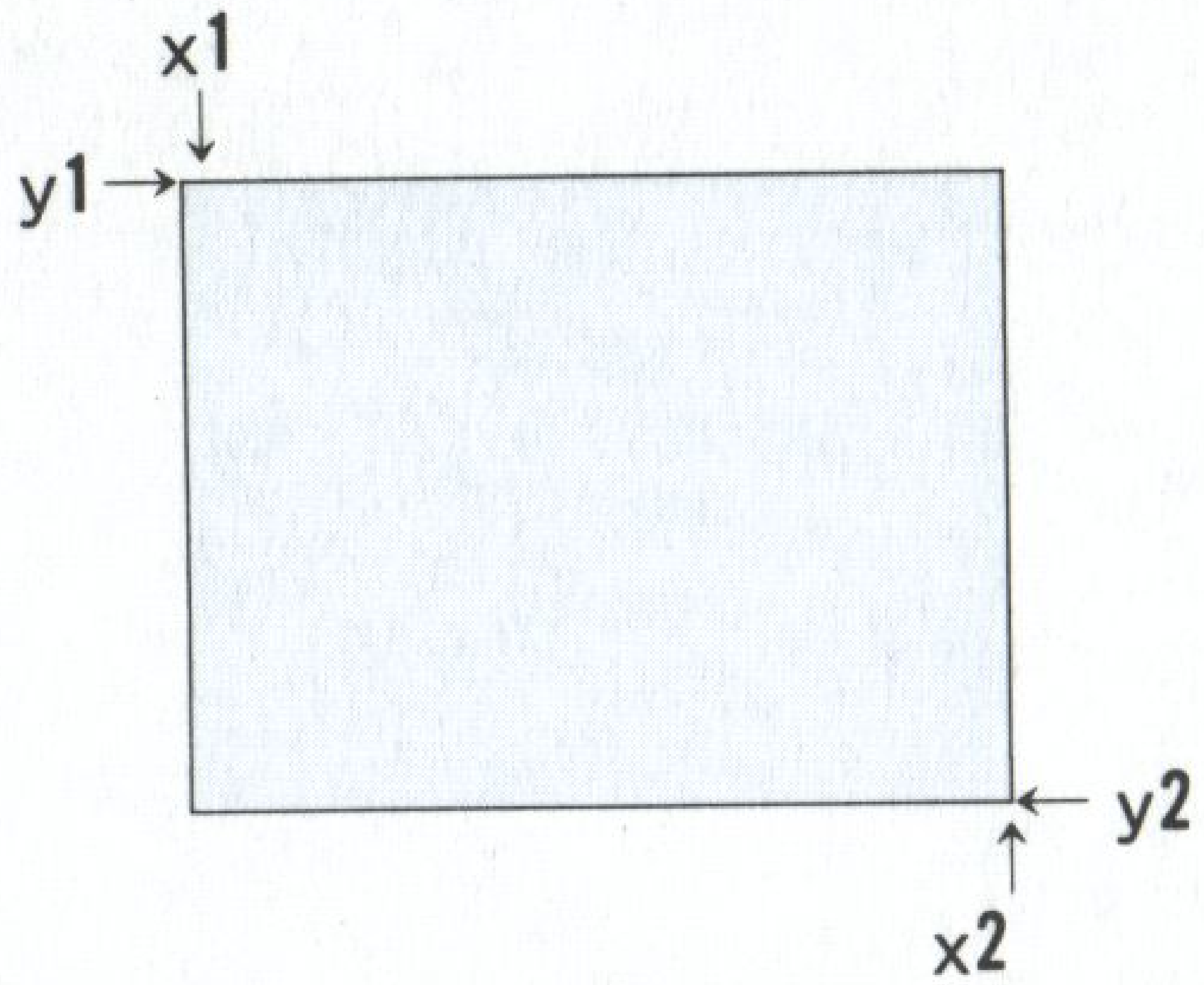
### ■機能

指定した物理領域(四角型)をグラフィック・ビュー・ポートとして定義する。

### ■書式

**VIEW**　[[**SCREEN**] (x1, y1) - (x2, y2)
[, [カラー] [, 境界線]]]

### ■引数とオプション

| 引数とオプション | 機　能 |
|---|---|
| SCREEN | このオプションを使うと、物理座標の指定は絶対座標で行なうことになる。 |
| x1, y1<br>x2, y2 | 定義したい物理座標を指定する。<br><br> |

| | |
|---|---|
| カラー | 定義した領域を特定のカラーで塗りつぶしたいときにそのコードを指定する。 |
| 境界線 | 定義した領域を特定のカラーで囲む境界線にカラーを設定したいときに、そのカラー・コードを指定する。 |

## ■文例

```
VIEW SCREEN (90, 90)-(310, 310), 1, 3
FOR aspect = 1 TO 5 STEP .2
   CIRCLE (200, 200), 100, 3, , , aspect
NEXT
   .
   .
```



## ■ノート

▶"カラー"を省略すると塗りつぶしは行なわれない。

▶"境界線"を省略すると境界線は引かれない。

# VIEW PRINT

## テキスト表示の行範囲の設定

### ■機能

スクリーンにおけるテキスト表示およびクリアの行範囲を設定する。スクロールは、この範囲で行なわれる

### ■書式

**VIEW PRINT ［上行番号 TO 下行番号］**

### ■オプション

| オプション | 機　能 |
| --- | --- |
| 上行番号と下行番号 | 設定したいテキスト表示の上限と下限の行を指定する。 |

### ■文例

```
VIEW PRINT 15 TO 23
```

### ■ノート

▶オプションを省略するとデフォルト値(上行番号=1、下行番号=25)に設定される。

▶スクロールは、その範囲で行なわれる。

▶このステートメントを実行したあとでのテキストの行座標指定は、相対的にではなく、スクリーンの最上行から数えた行番号で指定しなければならない。

# WAIT

## プログラムの実行の停止

### ■機能

入力ポートを監視中、プログラムの実行を停止させておく

### ■書式

**WAIT　ポート番号，AND 式［，XOR 式］**

### ■引数とオプション

| 引数とオプション | 機　能 |
|---|---|
| ポート番号 | 停止したいポートの番号(0～255)を指定する。 |
| AND 式 | AND 演算のためのビットパターンを指定する。 |
| XOR 式 | AND 演算の前に XOR 演算を行なうためのビットパターン指定する。 |

### ■文例

```
WAIT 10, 20
```

### ■ノート

▶演算結果が 0 であるあいだ、そのポートからの読み込みを続け、0 以外ならば、次のステートメントを実行する。

# WEND

## WHILE ステートメントへの戻り

### ■機能

対応する WHILE ステートメントに実行の流れを戻す

### ■書式

**WHILE　論理式**

⋮

**WEND**

### ■引数とオプション

なし

### ■文例（配列データを使ったバイナリー・サーチ）

```
SUB binarySearch (isStat, theArray(), keyData)
  low = LBOUND(theArray)
  high = UBOUND(theArray)
  WHILE (low <= high)
    mid = INT((low + high) / 2)
    IF (keyData < theArray(mid)) THEN
      high = mid - 1
    ELSEIF (keyData > theArray(mid)) THEN
      low = mid + 1
    ELSE
      isStat = mid
      EXIT SUB
    END IF
  WEND
  isStat = -1
END SUB
```

WEND

# WHILE

## 繰り返し実行

■機能

条件式が真(0以外)であるあいだ、対応するWENDステートメントとで囲んだブロック内のステートメントを実行する

■書式

**WHILE　論理式**
**⋮**
**WEND**

■引数

| 引数 | 機　能 |
|---|---|
| 論理式 | 繰り返し実行させたいときにテストする式である。 |

■文例（配列データを使ったバイナリー・サーチ）

```
SUB binarySearch (isStat, theArray(), keyData)
  low = LBOUND(theArray)
  high = UBOUND(theArray)
  WHILE (low <= high)
    mid = INT((low + high) / 2)
    IF (keyData < theArray(mid)) THEN
      high = mid - 1
    ELSEIF (keyData > theArray(mid)) THEN
      low = mid + 1
    ELSE
      isStat = mid
      EXIT SUB
    END IF
  WEND
  isStat = -1
END SUB
```

■ノート

▶論理式を省略することはできない。

# WIDTH

## 出力文字数の設定(Ver.4.5 拡張)

■機能

スクリーンにおけるテキスト表示の桁数と行数を設定する。

デバイスへ出力するときの1行の文字数を設定する。

プリンターの印刷文字数を設定する。

■書式

① WIDTH　[桁数][，行数]

② WIDTH　{#ファイル番号｜デバイス}，出力文字数

③ WIDTH　LPRINT　印刷文字数

■オプション(書式①)

| オプション | 機　　能 |
|---|---|
| 桁数 | 桁数は、80に設定されている。 |
| 行数 | 行数は、20または25に設定できる。 |

■オプション(書式②)

| 引数 | 機　　能 |
|---|---|
| #ファイル番号<br>または<br>デバイス名 | 出力したいデバイスに割り当てたファイル番号を指定する。<br>デバイス名("LPT1：")を指定する。 |
| 出力文字数 | 出力するときの1行の文字数を設定する。 |

WIDTH

■引数(書式③)

| 引数 | 機　能 |
|---|---|
| 印刷文字数 | 1行の印刷文字数を設定する。 |

■文例①

```
WIDTH 80, 25
```

■文例②

```
OPEN "LPT1:" FOR OUTPUT AS 1
WIDTH #1, 80
PRINT #1
   .
   .
```

■文例③

```
WIDTH LPRINT 80
```

■ノート

▶行数・桁数ともに変更できません。従来の BASIC との互換性を保つためにこのステートメントはある。

# WINDOW

## カレント・ビューポートの論理座標の定義

■機能

カレント・ビューポートの論理座標を独自に定義する

■書式

**WINDOW [[SCREEN] (x1, y1) - (x2, y2)]**

■オプション

| オプション | 機　能 |
|---|---|
| SCREEN | 指定するとスクリーンの上から下へ Y 座標が大きくなる。 |
| x1, y1<br>x2, y2 | スクリーンの論理座標を指定する。 |

■文例

```
READ x1, y1, x2, y2
DATA 1, 1, 100, 100
WINDOW (-640, -400)-(640, 400)
bytes = 4 + 4 * INT(((x2 - x1 + 1) + 7) / 8) * (y2 - y1 + 1)
DIM imageSample(bytes / 2 / 2) AS INTEGER
SCREEN 88, , 0, 1
LINE (x1, y1)-(x2, y2), , BF
GET (x1, y1)-(x2, y2), imageSample
FOR n = 1 TO 10
  PUT (n * 10, n * 10), imageSample, XOR
  PCOPY 0, 1
NEXT n
  .
  .
```

■ノート

▶オプションをすべて省略すると、デフォルトに設定される。

▶参照ステートメント：SCREEN、VIEW

▶参照関数：PMAP、POINT

# WRITE

## スクリーン表示



**■機能**

スクリーンにデータを表示する。シーケンシャル・ファイルへの書き出しイメージで表示する。

**■書式**

**WRITE ［式リスト］**

**■オプション**

| オプション | 機能 |
|---|---|
| 式リスト | 表示したい式のリストを指定する。 |

**■文例**

```
aName$ = "Quick"
WRITE aName$, "BASIC", 4.2
```

**<表示結果>**

```
"Quick","BASIC",4.2
```

**■ノート**

▶式は、","で区切る。この区切り記号も表示される。

▶式のなかの文字列データは、""" "で囲まれる。

# WRITE #

## シーケンシャル・ファイルへの書き出し

■機能

シーケンシャル・ファイルへデータを書き出す

■書式

**WRITE #ファイル番号，式リスト**

■引数

| 引数 | 機　　能 |
|---|---|
| #ファイル番号 | OPEN ステートメントでオープンしたときのファイル番号を指定する。 |
| 式リスト | 書き出したい式のリストを指定する。 |

■文例①

```
OPEN "Foods.Dat" FOR OUTPUT AS 1
DO
  INPUT ; "Product = ", aName$
  IF aName$ = "" THEN
    EXIT DO
  END IF
  INPUT " , Sales = ", sales
  WRITE #1, aName$, sales
LOOP
CLOSE 1
```

WRITE #

## ■文例②

```
TYPE FoodRecType
  itName AS STRING * 10
  amount AS INTEGER
  END TYPE
DIM fruit AS FoodRecType

OPEN "Fruit.Dat" FOR OUTPUT AS 1
DO
  INPUT ; " Fruit (<cr> for EXIT) = ", fruit.itName
  IF fruit.itName = SPACE$(10) THEN
    EXIT DO
  END IF
  INPUT " , amount = ", fruit.amount
  WRITE #1, fruit.itName, fruit.amount
LOOP
CLOSE 1
```

## ■ノート

▶書き込むファイルは、OUTPUTモードまたはAPPENDモードでオープンしなければならない。

▶書き込んだファイルは、必ずクローズしなければならない。

▶書き込むデータの行末には、改行コード(&H0D, &H0A)が付けられる。

▶フィールドは、", "で区切られる。

▶式のなかの文字列データは、""で囲まれる。

▶参照ステートメント：OPEN、INPUT #、CLOSE

# [第4章] 関数

この章では、QuickBASICが標準として提供している
関数のすべてをアルファベット順に説明します。
関数について機能別に知りたいときは、
付録の「機能別ステートメント・関数索引」を利用してください。

4

# この章の読み方

この章では、Quick BASIC(Ver.4.2)における全関数をアルファベット順に要約します。各関数は、以下のスタイルで記述します。

**関数名()**

機能についての要約が記述されています。

引数を必要とする関数には()がついています。

**■機能**

その関数の働きや用途についての要約が記述されています。

**■書式**

その関数についての完全な構文(シンタックス)が記述されています。

文字や記号は、以下のような用途で使います。

◉書式中の記号の記述と意味

| 用途 | 文字・記号 | 例 |
|---|---|---|
| そのまま書くべき箇所 | 大英文字および<br>, ” − ; / | CHR$( ) |
| 適当なものに置き換えて書くべき箇所 | 日本語の部分 | ファイル番号<br>コード1 |
| 省略可能な箇所 | [ ] | [, コード2 ...] |
| どれか1つを選ぶべき箇所 | { \| } | {数値式 \| 文字式} |
| 反復的に使える箇所 | ... | コード2 ... |

■引数と戻り値

引数および戻り値がまとめてあります。

◉引数および戻り値中の記号の記述と意味

| | |
|---|---|
| 引　数 | 関数に渡す引数のタイプや注意点について記述してあります。 |
| 戻り値 | 関数が返す値のタイプや注意点について記述してあります。 |

■文例

サンプルとしてその使用例を掲げてあります。

■ノート

パラメーターやオプションについての補足的な説明や使用するときの注意点について記述されています。

この章の読み方

# ABS()

## 絶対値

■機能

絶対値を求める

■書式

ABS(数値式)

■引数と戻り値

| | |
|---|---|
| 引　数 | 数値式…絶対値にしたい数値を渡す。 |
| 戻り値 | 符号のない数値が返される。 |

■文例

```
numericExpression = -4.2
absoluteValue = ABS(numericExpression)
PRINT absoluteValue, numericExpression
```

<実行結果>

```
 4.2              -4.2
```

■ノート

▶参照関数：SGN()

# ASC()

## ASCIIコード/漢字連続コード

**■機能**

文字の ASCII コードまたは漢字連続コードを知る

**■書式**

**ASC(文字式)**

**■引数と戻り値**

| | |
|---|---|
| 引　数 | 文字式…知りたい文字列を渡す。 |
| 戻り値 | 先頭文字が半角ならば、対応する ASCII コード(1～255)の整数が返される。<br>先頭文字が全角ならば、対応する漢字連続コードが返される。 |

**■文例**

```
' テキスト・ファイルをリストする
INPUT "Listing file name = ", source$
OPEN source$ FOR BINARY AS 1
ch$ = "": lineNo = 1
DO
  ch$ = INPUT$(1, 1)
  IF EOF(1) THEN
    EXIT DO
  ELSEIF ASC(ch$) <> &HD THEN
    PRINT ch$;
  END IF
LOOP
CLOSE 1
END
```

■ノート

▶文字式としてヌルを渡すと、ランタイム・エラーになる。

ERR No. メッセージ

5 引き数が許される範囲ではありません

▶先頭文字が全角のときに返される「漢字連続コード」とは、シフトJIS漢字コード表の2バイト文字である&H8140以降を、ASCIIコードに続くものとして256〜11535までのコードを連続的に付けたコードをいう。たとえば、つぎのようになっている。

| 全角 | 連続コード |
|---|---|
| 、 | 257 |
| 。 | 258 |
| ， | 259 |
| | ⋮ |

# ATN()

## 逆正接の計算

**■機能**

逆正接(アークタンジェント)を求める

**■書式**

**ATN(数値式)**

**■引数と戻り値**

| | |
|---|---|
| 引　数 | 数値式…ラジアンを渡す。 |
| 戻り値 | ラジアンの逆正接を返す。 |

**■文例**

```
CONST PI = 3.14593
PRINT CINT(ATN(1) * 180 / PI)
```

**<実行結果>**

```
45
```

**■ノート**

▶ラジアンは、$-\pi/2$～$\pi/2$の間でなければならない。

▶引数の型が倍精度型のときは、倍精度型の戻り値を返すが、それ以外は、単精度型の戻り値になる。

▶参照関数：TAN()

# CDBL()

## 倍精度型実数への変換

### ■機能

数値を倍精度型の実数に変換する

### ■書式

**CDBL(数値式)**

### ■引数と戻り値

| | |
|---|---|
| 引　数 | 数値式…倍精度型の実数に変換したい数値を渡す。 |
| 戻り値 | 倍精度型実数に変換した数値が返される。 |

### ■文例

```
numericExpression = 22 / 7
numericReturn# = CDBL(numericExpression)
PRINT numericReturn#; numericExpression; 22# / 7#
```

**<実行結果>**

```
3.142857074737549  3.142857  3.142857142857143
```

### ■ノート

▶数値の型を変換するけれども、引数自体を倍精度ベースで計算しないならば、返される数値は、引数よりも正確であることはない。

# CDBL$()

## 2バイト文字への変換

### ■機能

1バイト(半角)文字を2バイト(全角)文字に変換する

### ■書式

**CDBL$(文字列式)**

### ■引数と戻り値

| | |
|---|---|
| 引　数 | 文字列式…2バイト文字に変換したい文字列を渡す。 |
| 戻り値 | 1バイトの部分を2バイト文字にして返す。 |

### ■文例

```
stringExpression$ = "ﾏｲｸﾛｿﾌﾄ QuickBASIC4.2"
stringReturn$ = CDBL$(stringExpression$)
PRINT stringReturn$; SPC(2); stringExpression$
```

**<実行結果>**

```
マイクロソフト　ＱｕｉｃｋＢＡＳＩＣ４．２  ﾏｲｸﾛｿﾌﾄ QuickBASIC4.2
```

### ■ノート

▶ 2バイト文字に変換できる1バイト文字は、ASCIIコードの

&H20～&H7E

&HA1～&HDF

である(「**付録 ASCII コード表**」を参照のこと)。

CDBL$()

# CHR$()

## ASCIIコード/漢字連続コード対応文字

### ■機能

ASCIIコードおよび漢字連続コードに対応する文字を知る

### ■書式

**CHR$(コード1 [, コード2 ...])**

### ■引数と戻り値

| | |
|---|---|
| 引　数 | コード1　　…得たい文字のコードを渡す。<br>コード2以降…オプションの引数として渡すことができる。 |
| 戻り値 | 引数がASCIIコード(255以下)のとき、制御文字または1バイト文字が返される。<br>引数が漢字連続コード(256以上)のとき、全角文字(2バイト文字)が返される。 |

### ■文例

```
PRINT CHR$(65), "A", JIS$("A")
PRINT JIS$("、"), CHR$(257), CHR$(VAL("&J2122"))
PRINT CHR$(344, &H51, &H75, &H69, &H63, &H6B, 344)
```

**<実行結果>**

```
A            A            65
2122         、            、
☆Quick☆
```

■ノート

▶渡すことのできるコードは、以下のとおりである。

| コード | 10進数 | 8進数 | 16進数 |
|---|---|---|---|
| ASCIIコード | 0～　255 | &0　～　&377 | &H0　～　&HFF |
| 漢字連続コード | 256～11535 | &400～&26417 | &H100～&H2D0F |

CHRS

# CINT()

## 整数型数値への変換

**■機能**

数値を整数型の数値に変換する。小数点以下があるときは四捨五入してから整数型に変換する。

**■書式**

**CINT(数値式)**

**■引数と戻り値**

| | |
|---|---|
| 引　数 | 数値式…整数型にしたい数値を渡す。 |
| 戻り値 | 整数型に変換された数値が返される。 |

**■文例**

```
numericReturn1% = CINT(-41.54)
numericReturn2% = CINT(41.54)
PRINT numericReturn1%, numericReturn2%
```

**<実行結果>**

```
-42             42
```

**■ノート**

▶引数は、型に関係なく-32768〜32767の範囲内の数値でなければならない。

▶引数が実数ならば、小数第1位を四捨五入してから整数型の数値を返す。

# CLNG()

## 長整数型数値への変換

### ■機能

数値を長整数型の数値に変換する。小数点以下があるときは四捨五入してから長整数型に変換する。

### ■書式

**CLNG(数値式)**

### ■引数と戻り値

| | |
|---|---|
| 引　数 | 数値式…長整数型にしたい数値を渡す。 |
| 戻り値 | 長整数型に変換された数値が返される。 |

### ■文例

```
numericReturn1& = CLNG(-41999.54)
numericReturn2& = CLNG(41999.54)
PRINT numericReturn1&, numericReturn2&
```

**<実行結果>**

```
-42000        42000
```

### ■ノート

▶引数は、型に関係なく-2147483648～2147483647の範囲内の数値でなければならない。

▶引数が実数ならば、小数第１位を四捨五入してから長整数型の数値を返す。

# COMMAND$

## コマンド・ライン引数

■機能

MS-DOSのコマンド・ライン引数を得る

■書式

**COMMAND$**

■引数と戻り値

| | |
|---|---|
| 引　数 | なし |
| 戻り値 | コマンド・ラインの入力を1つの文字列として返す。 |

■文例

```
PRINT COMMAND$
```

<実行結果>

```
-L50 -W70
```

■ノート

▶ QuickBASICの環境下では、つぎのどちらかの方法で設定されているコマンド・ライン引数を返す。

※ QuickBASICの起動時に/CMDオプションを使って設定したコマンド・ライン引数

※"R/実行"メニューにおける"C/COMMAND$入力…" コマンドを使って設定したコマンド・ライン引数

▶実行可能ファイルの中では、そのファイルの起動時におけるコマンド・ライン引数を返す。

▶コマンド・ライン引数がないときは、ヌルを返す。

# COS()

## 余弦の計算

■機能

余弦(コサイン)を求める

■書式

COS(数値式)

■引数と戻り値

| | |
|---|---|
| 引　数 | 数値式…ラジアンを渡す。 |
| 戻り値 | ラジアンの余弦を返す。 |

■文例

```
CONST PI = 3.14593
angle = 60
PRINT COS(angle * PI / 180)
```

<実行結果>

```
 .4987474
```

■ノート

▶参照関数：SIN()、TAN()

# CSNG()

## 単精度型数値への変換

### ■機能

数値を単精度型の数値に変換する。小数点以下があるときは適当な位で四捨五入する。

### ■書式

**CSNG(数値式)**

### ■引数と戻り値

| | |
|---|---|
| 引　数 | 数値式…単精度型にしたい数値を渡す。 |
| 戻り値 | 単精度型に変換された数値が返される。 |

### ■文例

```
numericReturn! = CSNG(41999.77777777777#)
PRINT numericReturn!, 41999.77777777777#
```

**<実行結果>**

```
41999.78      41999.77777777777
```

### ■ノート

▶参照関数：CINT()

# CSNG$()

## 1バイト文字への変換(Ver.4.5 拡張)

**■機能**

2バイト(全角)文字を1バイト(半角)文字に変換する

**■書式**

**CSNG$(文字列式)**

**■引数と戻り値**

| | |
|---|---|
| 引　数 | 文字列式…1バイト文字に変換したい文字列を渡す。 |
| 戻り値 | 2バイトの部分を1バイト文字にして返す。 |

**■文例**

```
stringExpression$ = "まいぐろぞぶど　ぐいっぐぺーじっぐ"
stringReturn$ = CSNG$(stringExpression$)
PRINT stringReturn$
```

**<実行結果>**

```
マイクロソフト クイックヘーシック
```

**■ノート**

▶1バイト文字に変換できる2バイト文字は、変換後の1バイト文字がASCIIコードの&H20～&H7E　&HA1～&HDFになるものである。

▶ひらがな・濁音・破裂音は半角のカタカナに変換される(Ver.4.5のみ)。

# CSRLIN

## カーソルの行位置

### ■機能

カーソルが現在ある行位置を知る

### ■書式

**CSRLIN**

### ■引数と戻り値

| 引　数 | なし |
|---|---|
| 戻り値 | 現在のカーソルの行位置を整数で返す。 |

### ■文例

```
LOCATE 4, 2: PRINT "QuickBASIC"
numericReturn% = CSRLIN
PRINT numericReturn%
```

**<実行結果>**

```
QuickBASIC
5
```

### ■ノート

▶カーソルの行位置は、1〜25の整数で返される。

▶カーソルの桁位置を知るには、POS関数を使う。

# CVI() /CVL() /CVS() /CVD()

## 数値への変換

■機能

2バイト文字を整数型の数値に変換する

4バイト文字を長整数型の数値に変換する

4バイト文字を単精度型の数値に変換する

8バイト文字を倍精度型の数値に変換する

■書式

**CVI(2バイト文字)**

**CVL(4バイト文字)**

**CVS(4バイト文字)**

**CVD(8バイト文字)**

■引数と戻り値

| | |
|---|---|
| 引　数 | 変換したい型にあった関数にそれぞれの引数を渡す。<br>2バイト文字…整数型の数値に変換したい2バイト文字を渡す。<br>4バイト文字…長整数型の数値に変換したい4バイト文字を渡す。<br>4バイト文字…単精度型の数値に変換したい4バイト文字を渡す。<br>8バイト文字…倍精度型の数値に変換したい8バイト文字を渡す。 |
| 戻り値 | 変換された数値が返される。 |

■文例

```
OPEN "Fruit.Dat" FOR RANDOM AS 1 LEN = 12
FIELD #1, 10 AS nameFld$, 2 AS salesFld$
recNum = 1
PRINT " N A M E      amount "
PRINT "---------------------"
DO
  GET #1, recNum
  IF EOF(1) THEN
    EXIT DO
  END IF
  itName$ = nameFld$
  amount = CVI(salesFld$)
  PRINT itName$; " = "; USING "##,###"; amount
  recNum = recNum + 1
LOOP
CLOSE 1
END
```

■ノート

▶参照関数：MKI$、MKL$、MKS$、MKD$

# CVSMBF() /CVDMBF()

## IEEEフォーマット数値への変換

■機能

マイクロソフト・バイナリー・フォーマット(MBF)の4バイト文字を単精度型のIEEEフォーマットの数値に変換する。

マイクロソフト・バイナリー・フォーマット(MBF)の8バイト文字を倍精度型のIEEEフォーマットの数値に変換する。

■書式

**CVSMBF(4バイト文字)**

**CVDMBF(8バイト文字)**

■引数と戻り値

| | |
|---|---|
| 引　数 | 4バイト文字…単精度型IEEEフォーマットに変換したいMBF4バイト文字をCVSMBF関数に渡す。<br>8バイト文字…倍精度型IEEEフォーマットに変換したいMBF8バイト文字をCVDMBF関数に渡す。 |
| 戻り値 | 単精度型または倍精度型のIEEEフォーマット数値を返す。 |

■ノート

▶日付の設定は、DATEステートメントで行なえる。

■文例

```
' データ・ファイルの変換(MBFからIEEEへ)
TYPE MBFtype
  itName AS STRING * 10
  amount AS STRING * 4
  total  AS STRING * 8
  END TYPE
TYPE IEEEtype
  itName AS STRING * 10
  amount AS SINGLE
  total  AS DOUBLE
  END TYPE
DIM MBFrecord AS MBFtype, IEEErecord AS IEEEtype
OPEN "MBFFILE.dat" FOR RANDOM AS 1 LEN = LEN(MBFrecord)
OPEN "IEEEFILE.dat" FOR RANDOM AS 2 LEN = LEN(IEEErecord)
recNum = 1
DO
  GET #1, recNum, MBFrecord
  IF EOF(1) THEN
    EXIT DO
  ELSE
    IEEErecord.itName = MBFrecord.itName
    IEEErecord.amount = CVSMBF(MBFrecord.amount)
    IEEErecord.total = CVDMBF(MBFrecord.total)
    PUT #2, recNum, IEEErecord
  END IF
  recNum = recNum + 1
LOOP
CLOSE 1, 2
END
```

■ノート

▶「マイクロソフト・バイナリー・フォーマット(MBF)」とは、実数をバイナリー文字列形式で処理する方式をいい、単精度型の実数は4文字で、倍精度の実数は8文字の文字列で処理する。従来のBASICにおけるランダムアクセス・ファイルにおける実数は、この形式で保存されています。

▶「IEEEフォーマット」とは、実数をMBFよりもさらに精度の高い形式で処理する方式である。

▶ QuickBASICを起動するときおよびBCでコンパイルするときのオプションとして/MBFがある(「**Vol.1 4・1 Quick BASICの起動**」および「**Vol.1 9・3 コンパイル**」を参照)。

▶参照関数:CVSMBF()/CVDMBF()

# DATE$

## 現在の日付

**■機能**

現在のクロック日付を知る

**■書式**

**DATE$**

**■引数と戻り値**

| | |
|---|---|
| 引　数 | なし |
| 戻り値 | 現在のクロック日付をつぎの形式の文字列で返す。YYYY-MM-DD |

**■文例**

```
year$ = LEFT$(DATE$, 4)
month$ = MID$(DATE$, 6, 2)
day$ = RIGHT$(DATE$, 2)
PRINT "西暦 " + year$ + " 年 " + month$ + " 月 " + day$ + " 日"
```

**<実行結果>**

```
西暦 1989 年 06 月 14 日
```

DATE$

# ENVIRON$()

## 環境変数名と環境文字列

**■機能**

MS-DOS の環境変数名および環境文字列を知る

**■書式**

**ENVIRON$({環境変数名｜環境設定番号})**

**■引数と戻り値**

| | |
|---|---|
| 引　数 | つぎのどちらかを引数として渡さなければならない。<br>環境変数名　…知りたい設定内容に対応する環境変数名を文字列にして渡す。<br>環境設定番号…n 番目に設定されている環境変数名とその環境文字列の両方を知りたいときは、その番号を数値にして渡す。 |
| 戻り値 | 引数として環境変数名を渡すと、その設定内容である環境文字列だけが返される。<br>引数として環境設定番号を渡すと、その環境変数名とその設定内容である環境文字列が<br>　　環境変数名＝環境文字列<br>の文字列形式で返される。 |

■文例

```
SHELL "SET"
PRINT
PRINT ENVIRON$("PATH")
PRINT ENVIRON$("LIB")
PRINT ENVIRON$(1)
PRINT ENVIRON$(2)
PRINT ENVIRON$(3)
```

<実行結果>

```
COMSPEC=¥COMMAND.COM
PROMPT=$P$G
PATH=A:¥STUFF;A:¥DOS;A:¥QB¥BIN
LIB=A:¥QB¥LIB

A:¥STUFF;A:¥DOS;A:¥QB¥BIN
A:¥QB¥LIB
COMSPEC=¥COMMAND.COM
PROMPT=$P$G
PATH=A:¥STUFF;A:¥DOS;A:¥QB¥BIN
```

■ノート

▶引数である環境変数名は、大文字でなければならない。

▶引数である環境設定番号は、1～255の範囲内でなければならない。

▶引数に該当するものがないときは、ヌルが返される。

# EOF()

## ファイルの終わりのテスト

### ■機能

ファイルの終わり(END OF FILE)かどうかを調べる

### ■書式

**EOF(ファイル番号)**

### ■引数と戻り値

| | |
|---|---|
| 引数 | ファイル番号…オープンしたときに割り当てたファイル番号(整数型)を渡す。 |
| 戻り値 | ファイルの終わりのとき、-1(真)を返す。 |

### ■文例

```
TYPE FoodRecType
  itName AS STRING * 10
  amount AS INTEGER
  END TYPE
DIM fruit AS FoodRecType

OPEN "Fruit.Dat" FOR RANDOM AS 1 LEN = LEN(fruit)
recNum = 1
PRINT " N A M E      amount "
PRINT "--------------------"
DO
  GET #1, recNum, fruit
  IF EOF(1) THEN
    EXIT DO
  END IF
  PRINT fruit.itName; " = "; USING "##,###"; fruit.amount
  recNum = recNum + 1
LOOP
CLOSE 1
END
```

■ノート

▶オープンしたファイルの形態によって、ファイルの終わりかどうかを調べるタイミングが異なるので注意して使わなければならない。

| ファイルのタイプ | タイミング |
|---|---|
| シーケンシャル | 新しいデータを読み込む前にテストすること |
| ランダムアクセス | データを読み込んだ後でテストすること |
| バイナリー | データを読み込んだ後でテストすること |

▶参照ステートメント：OPEN、INPUT #、GET #

EOF ( )

# ERDEV/ERDEV$

## エラー・コードとデバイス名

**■機能**

致命的なデバイス・エラーが生じたときのエラー・コードとそのデバイス名を知る。

**■書式**

**ERDEV/ERDEV$**

**■引数と戻り値**

| 引　数 | なし |
|---|---|
| 戻り値 | ERDEV 関数は、致命的エラー・コードを返し、ERDEV$関数は、そのデバイス名を返す。 |

**■文例**

```
ON ERROR GOTO ErrorHandleRoutine
10 'FileOpen ...
  CLS
  OPEN "B:Fruit.dat" FOR RANDOM AS 1
  OPEN "B:Animal.dat" FOR RANDOM AS 2
20 'DateRead ...
        .
        .
        .
ErrorHandleRoutine:
   PRINT "Error Number = "; ERL
   PRINT "Error Code   = "; ERR
   PRINT "Error Device = "; ERDEV$
   PRINT "Error itCode = "; ERDEV MOD &H100
   RESET
   END
```

<実行結果>

```
Error Number =  10
Error Code   =  71
Error Device = B:
Error itCode =  2
```

■ノート

▶ ERDEV 関数は、DOS における割り込みタイプ24H のエラー処理が行なわれたときの DI におけるエラー・コード(下位バイト)とエラー属性ワード(上位バイト)を返す。

| コード | エラーの状態 |
|---|---|
| &H00 | ディスクがライト・プロテクトになっている |
| &H01 | ユニットが存在しない |
| &H02 | ドライブにディスクがセットされていない |
| &H03 | コマンドが存在しない |
| &H04 | CRC チェック・エラーが発生した |
| &H05 | 不適当なストラクチャー・サイズ |
| &H06 | アクセスできない |
| &H07 | デバイスが存在しない |
| &H08 | セクタが存在しない |
| &H09 | プリンターがペーパー・アウトになった |
| &H0A | 書き込みできない |
| &H0B | 読み込みできない |
| &H0C | ディスクが壊れている |

# ERL

## エラー発生の行番号

■機能

エラーが発生した行番号を知る

■書式

**ERL**

■引数と戻り値

| 引　数 | なし |
|---|---|
| 戻り値 | エラーが発生した行番号を返す。 |

■文例

```
100  ON ERROR GOTO ErrorHandleRoutine
110  FileOpen:
120    OPEN "Fruit.dat" FOR RANDOM AS 1
130    OPEN "Animal.dat" FOR RANDOM AS 1
140  DateRead:
 .
 .
 .
900  ErrorHandleRoutine:
910    PRINT "Error Number = "; ERL
920    PRINT "Error Code   = "; ERR
930    RESET
940    END
```

**<実行結果>**

```
Error Number =  130
Error Code   =  55
```

ERL

■ノート

▶ソース・プログラムに行番号が付いているときはその行番号を返し、付いていないときは0が返される。

▶参照ステートメント：ON ERROR

▶参照関数：ERR、ERDEV$、ERDEV

# ERR

## エラー・コード

### ■機能

エラーのコードを知る

### ■書式

**ERR**

### ■引数と戻り値

| | |
|---|---|
| 引　数 | なし |
| 戻り値 | 発生したエラーのコードを返す。 |

### ■文例

```
100  ON ERROR GOTO ErrorHandleRoutine
110  FileOpen:
120    OPEN "Fruit.dat" FOR RANDOM AS 1
130    OPEN "Animal.dat" FOR RANDOM AS 1
140  DateRead:
 .
 .
 .
900  ErrorHandleRoutine:
910    PRINT "Error Number = "; ERL
920    PRINT "Error Code   = "; ERR
930    RESET
940    END
```

**<実行結果>**

```
Error Number =  130
Error Code   =  55
```

### ■ノート

▶エラー・コードについては、「付録」を参照のこと。

▶参照ステートメント：ON ERROR

▶参照関数：ERL 、ERDEV$、ERDEV

# EXP()

## 指数関数の計算

### ■機能

自然対数の底e(2.71828...)のべき乗を計算する

### ■書式

**EXP( x )**

### ■引数と戻り値

| | |
|---|---|
| 引　数 | x…x乗したいときのxを渡す。 |
| 戻り値 | 計算した指数関数を返す。 |

### ■文例

```
x = 2
PRINT EXP(x)
```

**<実行結果>**

```
 7.389056
```

### ■ノート

▶参照関数：LOG()

# FILEATTR()

## ファイル・モード

■機能

オープンしているファイルのモードを知る

■書式

**FILEATTR(ファイル番号，種類)**

■引数と戻り値

| | |
|---|---|
| 引　数 | ファイル番号…オープンしたときに割り当てたファイル番号を渡す。<br>種類　　　　…知りたいモードの種類のコードを渡す。 |

| コード | 種類 |
|---|---|
| 1 | オープンしたときのモード |
| 2 | MS-DOS のファイル・ハンドラー |

| | |
|---|---|
| 戻り値 | 情報を数字コードで返す。<br>種類コードとして 1 を渡したときの戻り値は、つぎのようになっている。 |

| 戻り値 | ファイル・モード |
|---|---|
| 1 | INPUT(シーケンシャル) |
| 2 | OUTPUT(シーケンシャル) |
| 8 | APPEND(シーケンシャル) |
| 4 | RANDOM(ランダムアクセス) |
| 32 | BINARY(バイナリー) |

■文例

```
OPEN "Fruit.Dat" FOR RANDOM AS 1 LEN = LEN(fruit)
PRINT FILEATTR(1, 1), FILEATTR(1, 2)
CLOSE
```

<実行結果>

```
 4                5
```

■ノート

▶参照ステートメント：OPEN

# FIX()

## 小数点以下の切り捨て

■機能

小数点以下を切り捨て、整数部を取り出す

■書式

FIX(数値式)

■引数と戻り値

| | |
|---|---|
| 引　数 | 数値式…切り捨てたい数値を渡す。 |
| 戻り値 | 小数点以下を切り捨てて整数部だけを返す。 |

■文例

```
numericReturn1% = FIX(-41.54)
numericReturn2% = FIX(41.54)
PRINT numericReturn1%; numericReturn2%
```

<実行結果>

```
-41  41
```

■ノート

▶引数は、型に関係なく-32768～32767の範囲内の数値でなければならない。

▶ INT関数との違いは、引数xが正数のときは同じ値を返すが、xが負数のときにはINT関数はxよりも小さくxに最も近い整数を返すのに対し、FIX関数はxよりも大きくてxに最も近い整数を返す。

▶参照関数：INT()

# FRE()

## 利用可能なメモリー・サイズ

■機能

利用可能なメモリー・サイズを知る

■書式

FRE({数値式｜文字式})

■引数と戻り値

<table>
<tr><td>引　数</td><td>数値式…−1、−2、その他の数値を渡す。<br><table><tr><th>引数</th><th>メモリー</th></tr><tr><td>−1</td><td>数値型配列用のメモリー・サイズ</td></tr><tr><td>−2</td><td>スタック・サイズ</td></tr><tr><td>その他</td><td>文字列用のメモリー・サイズ</td></tr></table>文字式…任意の文字を渡すと文字列用のメモリー・サイズを知ることができる。</td></tr>
<tr><td>戻り値</td><td>利用可能なメモリー・サイズを返す。</td></tr>
</table>

■文例

```
PRINT FRE(-1), FRE(-2), FRE(1), FRE("dummy")
```

<実行結果>

```
144296  1152  46344  46344
```

■ノート

▶参照関数：SETMEM()

FRE()

# FREEFILE

## 使用可能なファイル番号

**■機能**

未使用のファイル番号を知る。これを使って次にオープンするファイルに新しいファイル番号を割り当てることができる。

**■書式**

**FREEFILE**

**■引数と戻り値**

| | |
|---|---|
| 引　数 | なし |
| 戻り値 | 使用可能なファイル番号を数値で返す。 |

**■文例**

```
TYPE FoodRecType
  itName AS STRING * 10
  amount AS INTEGER
  END TYPE
DIM fruit AS FoodRecType

FruitFileID = FREEFILE
OPEN "Fruit.Dat" FOR RANDOM AS FruitFileID LEN = LEN(fruit)
recNum = 1
PRINT " N A M E     amount "
PRINT "--------------------"
DO
  GET FruitFileID, recNum, fruit
  IF EOF(FruitFileID) THEN
    EXIT DO
  END IF
  PRINT fruit.itName; " = "; USING "##,###"; fruit.amount
  recNum = recNum + 1
LOOP
CLOSE FruitFileID
END
```

**<実行結果>**

```
 N A M E     amount
--------------------
APPLE       =   6,800
CHERRY      =   2,500
ORANGE      =   1,700
BANANA      =     480
MELON       =   7,000
   .
   .
   .
```

■ノート

▶参照ステートメント：OPEN、GET #、PUT #、WRITE、NPUT #、CLOSE

FREEFILE

# HEX$()

## 16進表現の文字列

**■機能**

数値を16進表現の文字列に変換する

**■書式**

**HEX$(数値式)**

**■引数と戻り値**

| | |
|---|---|
| 引数 | 数値式…16進表現の文字列にしたい数値を渡す |
| 戻り値 | 16進表現の文字列にして返す |

**■文例**

```
true = -1
PRINT HEX$(VARSEG(true)), HEX$(VARPTR(true))
```

**<実行結果>**

```
7787          44DA
```

**■ノート**

▶引数が実数ならば、小数第1位を四捨五入してから文字列に変換する。

▶戻り値の文字列には、先頭に&H は付かない。

# INKEY$

## キーボードからの1文字

**■機能**

キーボードから1文字を読み込む

**■書式**

**INKEY$**

**■引数と戻り値**

| 引　数 | なし |
|---|---|
| 戻り値 | 押されたキーの文字またはコードを文字列で返す。 |

**■文例**

```
CONST Null = ""
CLS
DO
  LOCATE 1, 1
  PRINT TIME$
  press$ = INKEY$
LOOP UNTIL press$ <> Null
IF (LEN(press$) = 2) THEN
  PRINT HEX$(ASC(LEFT$(press$, 1))) + HEX$(ASC(RIGHT$(press$, 1)))
ELSE
  PRINT HEX$(ASC(press$))
END IF
```

**<実行結果>**

```
19:01:44
1B
```

**■ノート**

▶何も押されないときは、ヌルが返される。

▶全角文字は、シフト JIS コードで返す。

▶キーボード上のキーは、キー・コードが返される(**付録**を参照のこと)

# INP()

## I/Oポートから1バイト

■**機能**

I/Oポートから1バイト読み込む

■**書式**

**INP(ポート番号)**

■**引数と戻り値**

| | |
|---|---|
| 引　数 | ポート番号…読み込みたいポートをオープンしたときに割り当てたファイル番号を渡す。 |
| 戻り値 | 読み込まれたデータが返される。 |

■**文例**

```
PRINT INP(2)
```

**<実行結果>**

```
 57
```

■**ノート**

▶参照ステートメント：OUT

INP()

# INPUT$()

## バイト単位でのファイル読み込み

**■機能**

指定したファイルから指定したバイト数だけ読み込む

**■書式**

**INPUT$(バイト数 [, [#] ファイル番号])**

**■引数と戻り値**

| | |
|---|---|
| 引　数 | バイト数　　…読み込みたいバイト数を渡す。<br>ファイル番号…読み込みたいファイルに割り当てたファイル番号を渡す。 |
| 戻り値 | 指定したバイト数を文字列として返す。 |

**■文例**

```
INPUT "Listing file name = ", source$

OPEN source$ FOR BINARY AS 1
ch$ = "": lineNo = 1
DO
  ch$ = INPUT$(1, 1)
  IF EOF(1) THEN
    EXIT DO
  ELSEIF ASC(ch$) <> &HD THEN
    PRINT ch$;
  END IF
LOOP
CLOSE 1
END
```

<実行結果>

```
Listing file name = README.DOC
    .
    .
    .
```

■ノート

▶読み込めるバイト数は、ファイルのタイプによって異なる。

| ファイルのタイプ | 読み込み可能なバイト数 |
|---|---|
| シーケンシャル<br>または<br>バイナリー | 32767バイト以下でなければならない。 |
| ランダムアクセス | 1レコード長以下でなければならない。 |

▶ファイル番号を省略したときは、標準入力(CON:)から読み込まれる。

▶参照関数:INPUT¥()、LOF()

# INPUT ¥()

## 文字数単位でのファイル読み込み

### ■機能

指定したファイルから指定した文字数だけ読み込む。半角・全角の区別なく読み込める。

### ■書式

**INPUT¥(文字数 [, [#] ファイル番号])**

### ■引数と戻り値

| | |
|---|---|
| 引　数 | 文字数　　　…読み込みたい文字数を渡す。<br>ファイル番号…読み込みたいファイルに割り当てたファイル番号を渡す。 |
| 戻り値 | 指定した文字数を読み込んで文字列として返す。 |

### ■文例

```
INPUT "Listing file name = ", source$

OPEN source$ FOR BINARY AS 1
ch$ = "": lineNo = 1
DO
  ch$ = INPUT¥(1, 1)
  IF EOF(1) THEN
    EXIT DO
  ELSEIF ASC(ch$) <> &HD THEN
    PRINT ch$;
  END IF
LOOP
CLOSE 1
END
```

**<実行結果>**

```
Listing file name = README.DOC
    .
    .
    .
```

■ノート

▶読み込めるバイト数は、ファイルのタイプによって異なる。

| ファイルのタイプ | 読み込み可能な文字数 |
|---|---|
| シーケンシャル<br>または<br>バイナリー | 32767バイト以下でなければならない。 |
| ランダムアクセス | 1レコード長以下でなければならない。 |

▶ファイル番号を省略したときは、標準入力(CON：)から読み込まれる。

▶参照関数：INPUT$()、LOF()

INPUT¥()

# INSTR()

## 文字列の位置

**■機能**

ある文字列が別の文字列のなかに存在するときのその位置を知る

**■書式**

**INSTR([開始位置], 文字式 1, 文字式 2)**

**■引数と戻り値**

| | |
|---|---|
| 引　数 | 開始位置…文字式1の中の何文字目以降の中を調べるかを指定する。<br>文字式 1…検索される文字列(この文字列の中を探す)<br>文字式 2…検索したい文字列(この文字列を探す) |
| 戻り値 | 文字式 1 の中に文字式 2 あるならば、文字式 2 の先頭文字が存在する位置を数値で返す。<br>存在しないならば、 0 を返す。 |

**■文例**

```
DECLARE SUB analyzePath (nest!, aDrv$, aDir$(), aPathName$)

 DIM dirNames$(1 TO 10)
 driveName$ = "": pathName$ = "A:¥QB¥PROGRAM¥DOSCALLS¥FILES"
 level = 0
 CALL analyzePath(level, driveName$, dirNames$(), pathName$)
 PRINT driveName$
 FOR i = 1 TO level
   PRINT dirNames$(i)
 NEXT i
 END
```

```
SUB analyzePath (nest, aDrv$, aDir$(), aPathName$)
  nest = 1: start = 1: position = 0
  DO
    position = INSTR(start, aPathName$, "¥")
    IF ((start = 1) AND (INSTR(aPathName$, ":") >= 1)) THEN
      aDrv$ = LEFT$(aPathName$, INSTR(aPathName$, ":"))
    ELSEIF ((start = 1) AND (INSTR(aPathName$, ":") = 0)) THEN
      aDrv$ = ""
    ELSEIF (position <> 0) THEN
      aDir$(nest) = MID$(aPathName$, start, position - start)
      nest = nest + 1
    ELSE
      aDir$(nest) = RIGHT$(aPathName$, LEN(aPathName$) - start + 1)
    END IF
    start = position + 1
  LOOP UNTIL (position = 0)
END SUB
```

**<実行結果>**

```
A:
QB
PROGRAM
DOSCALLS
FILES
```

■ノート

▶文字の位置が、バイト単位で数える。

▶参照関数：LEN()

INSTR()

# INT()

## 小数点以下の切り捨て

■機能

小数点以下を切り捨て、整数部を取り出す

■書式

INT(数値式)

■引数と戻り値

| | |
|---|---|
| 引　数 | 数値式…切り捨てたい数値を渡す。 |
| 戻り値 | 小数点以下を切り捨てて整数部だけを返す。 |

■文例

```
numericReturn1% = INT(-41.54)
numericReturn2% = INT(41.54)
PRINT numericReturn1%, numericReturn2%
```

<実行結果>

```
-42          41
```

■ノート

▶マイナスのときは、より小さい負の整数になる。

▶参照関数：FIX()、CINT()

# IOCTL$()

## デバイス・ドライバからの読み取り

**■機能**

デバイス・ドライバから制御データを読み取る。

**■書式**

**IOCTL$([#] ファイル番号)**

**■引数と戻り値**

| | |
|---|---|
| 引　数 | ファイル番号…デバイス・ドライバをオープンしたときに割り当てたファイル番号を渡す。 |
| 戻り値 | デバイス・ドライバからの制御文字データを返す。 |

**■文例**

```
OPEN "¥Driver¥HardCopy" OUTPUT AS 1
echo$ = IOCTL$(1)
```

**■ノート**

▶参照ステートメント：IOCTL

# JIS$()

## JISコード

**■機能**

文字のJISコードを知る。

**■書式**

**JIS$(文字式)**

**■引数と戻り値**

| | |
|---|---|
| 引　数 | 文字式…JISコードを知りたい文字または文字式を渡す。 |
| 戻り値 | 文字または文字式の先頭文字のJISコードを文字列で返す。 |

**■文例**

```
PRINT CHR$(65), "A", JIS$("A")
PRINT JIS$("、"), CHR$(257), CHR$(VAL("&J2122")), CHR$(VAL("&K0102"))
```

**<実行結果>**

```
A                    A                  65
2122                 、                 、                 、
```

**■ノート**

▶戻り値は、引数が1バイトならば10進形式で、2バイトならば16進形式で返される。

▶参照関数：KTN$、CHR$()

# KEXT$()

## 文字の抽出

**■機能**

文字列に含まれている文字の中から半角文字または全角文字のどちらかを抽出する。

**■書式**

**KEXT$(文字式，区分)**

**■引数と戻り値**

| | |
|---|---|
| 引　数 | 文字式…抽出したい文字が含まれる文字式を渡す。<br>区分　…半角だけを抽出したいときは0を、全角文字だけを抽出したいときは1を渡す。 |
| 戻り値 | 指定した区分に合う文字だけからなる文字列が返される。 |

**■文例**

```
hankaku = 0: zenkaku = 1
PRINT KEXT$("ＱuickＢASIC.2", zenkaku)
```

**<実行結果>**

```
ＱＢ
```

**■ノート**

▶該当する文字が含まれていないときは、ヌルが返される。

KEXT$()

# KINSTR()

## 文字位置

**■機能**

ある文字列が別の文字列のなかに存在するときのその位置を知る。半角・全角に関係なくそれらを1文字として数える。

**■書式**

**KINSTR([開始位置], 文字式1, 文字式2)**

**■引数と戻り値**

| | |
|---|---|
| 引　数 | 開始位置…文字式1の中の何文字目以降の中を調べるかを指定する。<br>文字式1…検索される文字列(この文字列の中を探す)<br>文字式2…検索したい文字列(この文字列を探す) |
| 戻り値 | 文字式1の中に文字式2あるならば、文字式2の先頭文字が存在する位置を数値で返す。<br>存在しないならば、0を返す。 |

**■文例**

```
position = KINSTR("QuickＢＡＳＩＣ4.2", "ＢＡＳＩＣ")
PRINT position
```

**<実行結果>**

```
6
```

**■ノート**

▶参照関数：INSTR()

# KLEN()

## 文字数

■**機能**

文字列の文字数を知る。半角も全角も１文字として数える。

■**書式**

**KLEN(文字式)**

■**引数と戻り値**

| | |
|---|---|
| 引　数 | 文字式…文字数を知りたい文字式を渡す。 |
| 戻り値 | 文字数が返される。 |

■**文例**

```
letterNum = KLEN("quickＢＡＳＩＣ4.2")
PRINT letterNum
```

**<実行結果>**

```
13
```

■**ノート**

▶参照関数：LEN()

# KMID$()

## 部分文字列の抽出

■機能

文字列の中の部分文字列を取り出す。

■書式

**KMID$(文字式，開始位置，[，文字数])**

■引数と戻り値

| | |
|---|---|
| 引　数 | 文字式　…元になる文字列を渡す。<br>開始位置…取り出す部分文字列が文字式の中で始まる位置を文字式の先頭から半角(1バイト)単位で数えて数値で渡す。<br>文字数　…取り出す文字数を文字式の先頭から半角・全角に関係なく数えた文字数を数値で渡す。 |
| 戻り値 | 引数の文字式の中の開始位置より文字数だけ取り出して文字列で返す。 |

■文例

```
language$ = KMID$("QuickＢＡＳＩＣ4.2", 6, 5)
PRINT language$
```

<実行結果>

```
ＢＡＳＩＣ
```

■ノート

▶引数である開始位置は、1～32767の範囲の数値でなければならない。

▶引数である文字数は、省略可能であるが指定するときは0～32767の範囲の数値でなければならない。

▶その開始位置および文字数が実数ならば、小数第1位で四捨五入される。

▶その開始位置が文字式の長さを超えているか、またはその文字数が0ならば、ヌルが返される。

▶引数である文字数を省略するか、またはその文字数が開始位置から末尾までの文字数を超えているならば、開始位置から末尾までの文字列が返される。

▶参照関数：KLEN()、KPOS()

KMID$()

# KPOS()

## 文字の位置

■機能

ある文字列においてn文字目の文字が何バイトにあるかを知る。

■書式

**KPOS(文字式, n)**

■引数と戻り値

| | |
|---|---|
| 引　数 | 文字式…検索の対象になる文字列を渡す。<br>n　…調べたい文字の文字位置が何文字目にあるかを数値で渡す（nは正数)。 |
| 戻り値 | n<=KLEN(文字式)ならば、先頭からのバイト位置を返す。<br>n>KLEN(文字式)ならば、0を返す。 |

■文例

```
PRINT KLEN("quickＢＡＳＩＣ4.2")
bytePos = KPOS("quickＢＡＳＩＣ4.2", 8)
PRINT bytePos
```

<実行結果>

```
13
10
```

■ノート

▶参照関数：KLEN()

# KTN$()

## 句点コード

■機能

文字の句点コードを知る。

■書式

**KTN$(文字式)**

■引数と戻り値

| | |
|---|---|
| 引　数 | 文字式…句点コードを知りたい文字または文字式を渡す。 |
| 戻り値 | 文字または文字式の先頭文字の句点コードを文字列で返す。 |

■文例

```
PRINT CHR$(65), "A", JIS$("A"), KTN$("A")
PRINT KTN$("、"), CHR$(257), CHR$(VAL("&K0102"))
```

<実行結果>

```
A               A               65              65
0102            、              、
```

■ノート

▶参照関数：JIS$、CHR$()

# LBOUND()

## 添字の下限

**■機能**

配列における添字の下限を知る

**■書式**

**LBOUND(配列名 [, 次元])**

**■引数と戻り値**

| | |
|---|---|
| 引　数 | 配列名…下限を知りたい配列の名前を渡す。<br>次元　…下限を知りたい配列の次元を渡す。 |
| 戻り値 | 指定した配列の次元の下限を返す。 |

**■文例**

```
DIM matrix(1980 TO 1999, 1 TO 12)
PRINT LBOUND(matrix), LBOUND(matrix, 2)
```

**<実行結果>**

```
1980            1
```

**■ノート**

▶次元を省略すると、1次元を指定したことになる。

▶参照ステートメント：DIM

▶参照関数：UBOUND()

# LCASE$()

## 小文字への変換

**■機能**

文字列におけるアルファベットの大文字部分を小文字に変換する

**■書式**

**LCASE$(文字式)**

**■引数と戻り値**

| | |
|---|---|
| 引　数 | 文字式…小文字に変換したい大文字を含む文字式を渡す。 |
| 戻り値 | 大文字の部分を小文字にした文字列を返す。 |

**■文例**

```
PRINT LCASE$("QuickＢａｓｉｃ4.2")
```

**<実行結果>**

```
quickｂａｓｉｃ4.2
```

**■ノート**

▶アルファベットの大文字として、

半角のA～Z

全角のＡ～Ｚ

のどちらでもよい。

▶参照関数：UCASE$()

# LEFT$()

## 文字列の左端部分の抽出

■機能

文字列の左端部分を取り出す

■書式

**LEFT$(文字式, 文字数)**

■引数と戻り値

| | |
|---|---|
| 引数 | 文字式…元になる文字列を渡す。<br>文字数…取り出す文字数を文字式の先頭から半角(1バイト)単位で数えて数値で渡す。 |
| 戻り値 | 引数の文字式の先頭から文字数だけ取り出して、文字列で返す。 |

■文例

```
quick$ = LEFT$("QuickBASIC4.2", 5)
PRINT quick$
```

<実行結果>

```
Quick
```

■ノート

▶引数である文字数は、0〜32767の範囲の数値でなければならない。

▶その文字数が実数ならば、小数第1位で四捨五入される。

▶その文字数が0ならば、ヌルが返される。

▶その文字数が文字式の長さを超えるならば，文字式と同じ文字列が返される。

▶参照関数：MID$、RIGHT$、INSTR

# LEN()

## 文字列の長さ/変数のバイト・サイズ

■機能

文字列のバイト長または変数のバイト・サイズを知る

■書式

**LEN({文字式｜変数名})**

■引数と戻り値

| | |
|---|---|
| 引　数 | 文字式…バイト長を知りたい文字式を渡す。<br>変数名…バイト・サイズを知りたい変数の名前を渡す。 |
| 戻り値 | バイト数でその長さまたはサイズが返される。 |

■文例

```
TYPE userType
  userName AS STRING * 10
  year     AS INTEGER
  month    AS INTEGER
  day      AS INTEGER
END TYPE
DIM userInfo AS userType
sizeOf = LEN(userInfo)
PRINT sizeOf
```

<実行結果>

```
16
```

■ノート

▶引数として変数名を使うときは、文字列型でなくてもよい。

▶参照ステートメント：OPEN

▶参照関数：KLEN()

# LOC()

## ファイル上の現在位置

**■機能**

ファイル上における現在の位置(最後にアクセスした位置)を知る

**■書式**

**LOC(ファイル番号)**

**■引数と戻り値**

| | |
|---|---|
| 引　数 | ファイル番号…知りたいファイルのオープン時のファイル番号を渡す。 |
| 戻り値 | ファイル内に位置を数値で返す。 |

**■文例**

```
TYPE FoodRecType
  itName AS STRING * 10
  amount AS INTEGER
  END TYPE
DIM fruit AS FoodRecType

FruitFileID = FREEFILE
OPEN "Fruit.Dat" FOR RANDOM AS FruitFileID LEN = LEN(fruit)
PRINT " N A M E     amount "
PRINT "--------------------"
DO
  GET FruitFileID, LOC(FruitFileID) + 1, fruit
  IF EOF(FruitFileID) THEN
    EXIT DO
  END IF
  PRINT fruit.itName; " = "; USING "##,###"; fruit.amount
LOOP
CLOSE FruitFileID
END
```

■ノート

▶オープンしたファイルの形態によって、ファイル内の位置は異なるので注意して使わなければならない。

| ファイルのタイプ | 位置 |
|---|---|
| シーケンシャル | オープン後最後にアクセスしたバイト位置/128を返す。 |
| ランダムアクセス | オープン後最後にアクセスしたレコード番号を返す。オープン直後は0を返す。 |
| バイナリー | オープン後最後にアクセスしたバイト位置を返す。オープン直後は0を返す。 |

▶参照ステートメント：OPEN、INPUT #、GET #

▶参照関数：LOF()、SEEK()

LOC()

# LOF()

## ファイルのバイト・サイズ

■機能

オープンしてあるファイルのバイト・サイズを知る

■書式

**LOF(ファイル番号)**

■引数と戻り値

| | |
|---|---|
| 引　数 | ファイル番号…知りたいファイルのオープン時のファイル番号を渡す。 |
| 戻り値 | ファイルのサイズをバイト数で返す。 |

■文例

```
TYPE FoodRecType
  itName AS STRING * 10
  amount AS INTEGER
  END TYPE
DIM fruit AS FoodRecType

OPEN "Fruit.Dat" FOR RANDOM AS 1 LEN = LEN(fruit)
recMax = LOF(1) / LEN(fruit)
enterPrompt$ = "Rec. No (1 -" + STR$(recMax) + ") "
   .
   .
   .
```

■ノート

▶参照ステートメント：OPEN、SEEK

# LOG()

## 自然対数の計算

■機能

eを底としたときのnの対数(自然対数)を求める

■書式

LOG(数値式)

■引数と戻り値

| | |
|---|---|
| 引　数 | n…自然対数をもとめるときのnを渡す。 |
| 戻り値 | nの自然対数を返す。 |

■文例

```
n = 2
PRINT LOG(n)
```

<実行結果>

```
.6931472
```

■ノート

▶参照関数：EXP()

LOG()

# LPOS()

## プリンター・ヘッドの現在行

■機能

プリンター・バッファ内のプリンター・ヘッドの現在行を知る

■書式

**LPOS(番号)**

■引数と戻り値

| | |
|---|---|
| 引　数 | 番号… プリンターのデバイス番号（LPT1:ならば１）を渡す。 |
| 戻り値 | プリンター・ヘッドがある現在の行を返す。 |

■文例

```
PRINT LPOS(1)
```

<実行結果>

```
 1
```

# LTRIM$()

## 先頭空白文字の除去

■機能

文字列の先頭にある空白文字を取り除く

■書式

LTRIM$(文字式)

■引数と戻り値

| | |
|---|---|
| 引　数 | 文字式…先頭にある空白文字を取り除きたい文字列を渡す。 |
| 戻り値 | 先頭にある半角および全角の空白文字を取り除いた文字列を返す。 |

■文例

```
stringExpression$ = "   QuickBASIC 4.2"
stringReturn$ = LTRIM$(stringExpression$)
PRINT stringExpression$, LEN(stringExpression$)
PRINT stringReturn$, LEN(stringReturn$)
```

<実行結果>

```
   QuickBASIC 4.2            17
QuickBASIC 4.2               14
```

■ノート

▶参照関数：RTRIM$()

# MID$()

## 部分文字列の抽出

**■機能**

文字列の中の部分文字列を取り出す

**■書式**

**MID$(文字式，開始位置［，文字数］)**

**■引数と戻り値**

| | |
|---|---|
| 引　数 | 文字式　…元になる文字列を渡す。<br>開始位置…取り出す部分文字列が文字式の中で始まる位置を文字式の先頭から半角(1バイト)単位で数えて数値で渡す。<br>文字数　…取り出す文字数を文字式の先頭から半角(1バイト)単位で数えて数値で渡す。 |
| 戻り値 | 引数の文字式の中の開始位置より文字数だけ取り出して文字列で返す。 |

**■文例**

```
language$ = MID$("QuickBASIC4.2", 6, 5)
PRINT language$
```

**<実行結果>**

```
BASIC
```

■ノート

▶引数である開始位置は，1〜32767の範囲の数値でなければならない。

▶引数である文字数は、省略可能であるが指定するときは0〜327671の範囲の数値でなければならない。

▶その開始位置および文字数が実数ならば、小数第1位で四捨五入される。

▶その開始位置が文字式の長さを超えているか、またはその文字数が0ならば、ヌルが返される。

▶引数である文字数を省略するか、またはその文字数が開始位置から末尾まで文字数を超えているならば、開始位置から末尾までの文字列が返される。

▶参照関数：LEFT$、RIGHT$、INSTR

# MKI$() /MKL$() /MKS$() /MKD$()

## 文字列への変換

■機能

整数型の数値を2バイト文字列に変換する

長整数型の数値を4バイト文字列に変換する

単精度型の数値を4バイト文字列に変換する

倍精度型の数値を8バイト文字列に変換する

■書式

**MKI$(整数型数値式)**

**MKL$(長整数型数値式)**

**MKS$(単精度型数値式)**

**MKD$(倍精度型数値式)**

■引数と戻り値

| | |
|---|---|
| 引　数 | 変換したい型にあった関数にそれぞれの引数を渡す<br>整数型数値式　…変換したい整数の式を渡す<br>長整数型数値式…変換したい長整数の式を渡す<br>単精度型数値式…変換したい単精度の式を渡す<br>倍精度型数値式…変換したい倍精度の式を渡す |
| 戻り値 | 変換された文字列が各バイト数で返される。 |

■文例

```
FIELD #1, 2 AS numFld$
num = 28
RSET numFld$ = MKI$(num)
```

```
FIELD #1, 4 AS numFld$
num = 12345
RSET numFld$ = MKL$(num)
```

```
FIELD #1, 4 AS numFld$
num = 721600
RSET numFld$ = MKS$(num)
```

```
FIELD #1, 8 AS numFld$
num = 12345000
RSET numFld$ = MKD$(num)
```

■ノート

▶参照関数：CVI()/CVL()/CVS()/CVD()

# MKSMBF$() /MKDMBF$()

## MBFフォーマット文字列への変換

■機能

単精度型のIEEEフォーマットの数値をマイクロソフト・バイナリー・フォーマット(MBF)の4バイト文字列に変換する。

倍精度型のIEEEフォーマットの数値をマイクロソフト・バイナリー・フォーマット(MBF)の8バイト文字に変換する。

■書式

**MKSMBF$(単精度型数値式)**
**MKDMBF$(倍精度型数値式)**

■引数と戻り値

| | |
|---|---|
| 引　数 | 単精度型数値式…MBF4バイト文字に変換したい単精度型IEEEフォーマットの数値式をMKSMBF$関数に渡す。<br>倍精度型数値式…MBF8バイト文字に変換したい倍精度型IEEEフォーマットの数値式をMKDMBF$関数に渡す。 |
| 戻り値 | 4バイトまたは8バイトのMBF文字列を返す。 |

## ■文例

```
' データ・ファイルの変換（IEEEからMBFへ）
TYPE MBFtype
  itName AS STRING * 10
  amount AS STRING * 4
  total  AS STRING * 8
  END TYPE
TYPE IEEEtype
  itName AS STRING * 10
  amount AS SINGLE
  total  AS DOUBLE
  END TYPE
DIM MBFrecord AS MBFtype, IEEErecord AS IEEEtype

OPEN "MBFFILE.dat" FOR RANDOM AS 1 LEN = LEN(MBFrecord)
OPEN "IEEEFILE.dat" FOR RANDOM AS 2 LEN = LEN(IEEErecord)
recNum = 1
DO
  GET #2, recNum, IEEErecord
  IF EOF(1) THEN
    EXIT DO
  ELSE
    MBFrecord.itName = IEEErecord.itName
    MBFrecord.amount = MKSMBF$(IEEErecord.amount)
    MBFrecord.total =  MKDMBF$(IEEErecord.total)
    PUT #1, recNum, MBFrecord
  END IF
  recNum = recNum + 1
LOOP
CLOSE 1, 2
END
```

## ■ノート

▶「IEEE フォーマット」および「マイクロソフト・バイナリー・フォーマット(MBF)」については、CVSMBF 関数/CVDMBF 関数を参照してください。

# OCT$()

## 8進表現文字列への変換

**■機能**

数値を8進表現の文字列に変換する

**■書式**

**OCT$(数値式)**

**■引数と戻り値**

| | |
|---|---|
| 引　数 | 数値式…8進表現の文字列にしたい数値を渡す。 |
| 戻り値 | 8進表現の文字列にして返す。 |

**■文例**

```
numericExpression = 255
stringReturn$ = OCT$(numericExpression)
PRINT stringReturn$, numericExpression
```

**<実行結果>**

```
377 255
```

**■ノート**

▶引数が実数ならば、小数第1位を四捨五入してから文字列に変換する。

▶戻り値の文字列には、先頭に&O は付かない。

# PEEK()

## メモリー・アドレスのストア内容

■機能

特定のメモリー・アドレスにストアされている内容を知る

■書式

**PEEK(アドレス)**

■引数と戻り値

| | |
|---|---|
| 引　数 | アドレス…読み込みたいアドレスのオフセットを渡す。 |
| 戻り値 | 現在のセグメントからのオフセット・アドレスにある内容をASCIIコードで返す。 |

■文例

```
language$ = "QuickBASIC"
cnt = 0
DO
  PRINT CHR$(PEEK(SADD(language$) + cnt));
  cnt = cnt + 1
LOOP WHILE (cnt < LEN(language$))
```

<実行結果>

```
QuickBASIC
```

■ノート

▶参照ステートメント：DEF　SEG

▶参照関数：POKE()

# PMAP()

## 物理座標/論理座標

**■機能**

論理座標に対応する物理座標、もしくは物理座標に対応する論理座標に変換する。

**■書式**

**PMAP(座標，変換方法)**

**■引数と戻り値**

<table>
<tr><td>引　数</td><td>座標　　…変換したい座標を渡す。<br>変換方法…どのように変換するのか指示するために次の番号を渡す。<br>
<table>
<tr><th>番号</th><th>変換の仕方</th></tr>
<tr><td>0</td><td>論理座標を物理 x に変換する</td></tr>
<tr><td>1</td><td>論理座標を物理座標 y に変換する</td></tr>
<tr><td>2</td><td>物理座標を論理座標 x に変換する</td></tr>
<tr><td>3</td><td>物理座標を論理座標 y に変換する</td></tr>
</table></td></tr>
<tr><td>戻り値</td><td>引数の座標を変換した別の座標を返す。</td></tr>
</table>

**■文例**

```
WINDOW (1, 1)-(640, 400)
x1 = 100: y1 = 100: x2 = 200: y2 = 200
LINE (x1, y1)-(x2, y2)
a1 = PMAP(x1, 0)
b1 = PMAP(y1, 1)
a2 = PMAP(x2, 2)
b2 = PMAP(y2, 3)
LINE (a1, b1)-(a2, b2)
```

**■ノート**

▶参照ステートメント：WINDOW

PMAP()

# POINT()

## ピクセルのカラー番号/座標

■機能

任意のピクセルのカラー番号、現在のグラフィック・カーソルの座標を知る

■書式

**POINT({x座標, y座標|座標指定})**

■引数と戻り値

| | |
|---|---|
| 引　数 | x座標、y座標…カラーを知りたい座標を渡す。<br>座標指定　　…どの座標を知りたいかを番号で渡す。<br><table><tr><th>番号</th><th>座標</th></tr><tr><td>0</td><td>物理座標　x</td></tr><tr><td>1</td><td>物理座標　y</td></tr><tr><td>2</td><td>論理座標　x</td></tr><tr><td>3</td><td>論理座標　y</td></tr></table> |
| 戻り値 | 座標を渡したときはそのカラー属性コードを、また座標指定を渡したときは現在の座標を返す。 |

■文例

```
WINDOW (-100, -100)-(540, 300)
x1 = 50: y1 = 50: x2 = 100: y2 = 250
LINE (x1, y1)-(x2, y2)
PRINT POINT(0); POINT(1); POINT(2); POINT(3)
PRINT POINT(POINT(2), POINT(3))
```

<実行結果>

```
200  50  100  250
7
```

■ノート

▶参照ステートメント：WINDOW、COLOR

# POS()

## カーソルの桁位置

**■機能**

カーソルが現在ある桁位置を知る

**■書式**

**POS(ダミー)**

**■引数と戻り値**

| | |
|---|---|
| 引　数 | ダミー…意味のない数値を引数として渡す |
| 戻り値 | 現在のカーソルの桁位置を整数で返す |

**■文例**

```
LOCATE 7, 2: PRINT "QuickBASIC";
numericReturn% = POS(0)
PRINT : PRINT numericReturn%
```

**<実行結果>**

```
 12
```

**■ノート**

▶ダミーの数値は、-32768〜32767の範囲の数値でなければならない。

▶カーソルの桁位置は、1〜80の整数で返される。

▶カーソルの行位置を知るには、CSRLIN 関数を使う。

# RIGHT$()

## 文字列の右端部分の抽出

### ■機能

文字列の右端部分を取り出す

### ■書式

**RIGHT$(文字式, 文字数)**

### ■引数と戻り値

| | |
|---|---|
| 引　数 | 文字式…末尾にある取り除きたい空白文字列を渡す。<br>文字数…取り出す文字数を文字式の先頭から半角(1バイト)単位で数えて数値で渡す。 |
| 戻り値 | 引数の文字式の末尾から文字数だけ取り出して文字列で返す。 |

### ■文例

```
versionNo$ = RIGHT$("QuickBASIC4.2", 3)
PRINT versionNo$
```

**<実行結果>**

```
4.2
```

### ■ノート

▶引数である文字数は、0～32767の範囲の数値でなければならない

▶その文字数が実数ならば、小数第1位で四捨五入される。

▶その文字数が0ならば、ヌルが返される。

▶その文字数が文字式の長さを超えるならば、文字式と同じ文字列が返される。

▶参照関数：LEFT$、MID$、INSTR

# RND()

## 乱数の発生

### ■機能

0から1の間の乱数を発生させる

### ■書式

**RND［(n)］**

### ■引数と戻り値

| | |
|---|---|
| 引　数 | n…乱数の発生の仕方を数字で渡す。 |
| 戻り値 | 0～1の単精度型数値を返す。 |

| n | 発生の仕方 |
|---|---|
| 正 | 乱数系列をベース発生させる。 |
| 0 | 前の発生した乱数を返す。 |
| 負 | 一定の値を発生させる。 |

### ■文例

```
' plotting
CLS
x1 = 200: y1 = 100: x2 = 400: y2 = 300
h = 0: v = 0: clr = 0
LINE (x1, y1)-(x2, y2), , BF
FOR n = 1 TO 10000
   h = INT((x2 - x1 + 1) * RND + x1)
   v = INT((y2 - y1 + 1) * RND + y1)
   clr = INT(RND * 7) + 1
   IF h < x1 OR h > x2 THEN
      LOCATE 1, 1: PRINT h
   END IF
   PSET (h, v), clr
NEXT n
```

### ■ノート

▶ nを省略すると正数が渡されたのと同じになる。

▶ RND()は、RANDOMIZEで指定したシード(乱数系列)をもとに発生される。

▶参照ステートメント：RANDOMIZE

# RTRIM$()

## 文字列の末尾空白文字の除去

■機能

文字列の末尾にある空白文字を取り除く

■書式

**RTRIM$(文字式)**

■引数と戻り値

| | |
|---|---|
| 引　数 | 文字式…末尾にある取り除きたい空白文字列を渡す。 |
| 戻り値 | 末尾にある半角と全角の空白文字を取り除いた文字列を返す。 |

■文例

```
stringExpression$ = "QuickBASIC 4.2   "
stringReturn$ = RTRIM$(stringExpression$)
PRINT stringExpression$, LEN(stringExpression$)
PRINT stringReturn$, LEN(stringReturn$)
```

<実行結果>

```
QuickBASIC 4.2                 17
QuickBASIC 4.2                 14
```

■ノート

▶参照関数：LTRIM$()

# SADD()

## 文字列の先頭アドレス

### ■機能

文字列の先頭アドレス(オフセット値)を知る

### ■書式

**SADD(文字列変数名)**

### ■引数と戻り値

| | |
|---|---|
| 引　数 | 文字列変数名…文字列型の変数名または配列要素を渡す。 |
| 戻り値 | そのとき現在のセグメント・アドレスからのオフセットを整数で返す。 |

### ■文例

```
language$ = "QuickBASIC"
cnt = 0
DO
  PRINT CHR$(PEEK(SADD(language$) + cnt));
  cnt = cnt + 1
LOOP WHILE (cnt < LEN(language$))
```

**<実行結果>**

```
QuickBASIC
```

### ■ノート

▶引数として固定長文字列変数名を渡すことはできない。

▶ SADD関数は、VARPTR関数を使って表わすことができる。

PEEK(VARPTR(varName$) + 3) * &H100 + PEEK(VARPTR(varName$) + 2)

▶参照関数：VARSEG()、VARPTR()、VARPTR$()

# SCREEN()

## 表示文字のASCIIコードとカラー属性(Ver.4.5 拡張)

### ■機能

テキスト・スクリーンに表示されている文字のASCIIコードおよびカラー属性を知る

### ■書式

**SCREEN(行, 桁 [, カラーフラグ])**

### ■引数と戻り値

| | |
|---|---|
| 引　数 | 行, 桁　　　…知りたい文字のテキスト・スクリーン座標を渡す。<br>カラーフラグ…カラー・コードを知るには0以外の数値を渡す。 |
| 戻り値 | 指定した座標に表示されている文字のASCIIコードまたはカラー・コードを返す。 |

### ■文例

```
CLS
COLOR 11
LOCATE 12, 20
PRINT "クイックベーシック"
COLOR 3
FOR c = 20 TO 38 STEP 2
  CH1$ = CHR$(SCREEN(12, c))
  CH2$ = CHR$(ABS(SCREEN(12, c + 1)))
  LOCATE 11, c: PRINT CH1$
  LOCATE 13, c: PRINT CH2$
NEXT c
```

**<実行結果>**

```
クイックベーシック
クイックベーシック
クイックベーシック
```

### ■ノート

▶ 2バイト文字のとき、1バイト目ならば正数の漢字連続コードが返され、2バイト目ならば負数の漢字連続コードが返される。

SCREEN()

# SEEK()

## ファイル内の次の位置

### ■機能

ファイル上の次にアクセスする位置を知る

### ■書式

**SEEK(ファイル番号)**

### ■引数と戻り値

| | |
|---|---|
| 引　数 | ファイル番号…知りたいファイルのオープン時のファイル番号を渡す。 |
| 戻り値 | ファイル上の次の位置を数値で返す。 |

### ■文例

```
TYPE FoodRecType
  itName AS STRING * 10
  amount AS INTEGER
  END TYPE
DIM fruit AS FoodRecType
FruitFileID = FREEFILE
OPEN "Fruit.Dat" FOR RANDOM AS FruitFileID LEN = LEN(fruit)
PRINT " N A M E     amount "
PRINT "--------------------"
DO
  GET FruitFileID, SEEK(FruitFileID), fruit
  IF EOF(FruitFileID) THEN
    EXIT DO
  END IF
  PRINT fruit.itName; " = "; USING "##,###"; fruit.amount
LOOP
CLOSE FruitFileID
END
```

■ノート

▶オープンしたファイルの形態によって、ファイル内の次の位置は異なるので注意して使わなければならない。

| ファイルのタイプ | 位置 |
|---|---|
| シーケンシャル | オープン後最後にアクセスしたバイト位置+1を返す。 |
| ランダムアクセス | オープン後最後にアクセスしたレコード番号+1を返す。オープン直後は1を返す。 |
| バイナリー | オープン後最後にアクセスしたバイト位置+1を返す。オープン直後は1を返す。 |

▶参照ステートメント：OPEN、INPUT #、GET #

▶参照関数：LOF()、LOC()

SEEK()

# SETMEM()

## farヒープのメモリー・サイズの変更

■機能

farヒープによって使われるメモリー・サイズをそのときのサイズから相対的に大きくしたり小さくできる

■書式

**SETMEM(バイト数)**

■引数と戻り値

| | |
|---|---|
| 引　数 | バイト数…増加したいときは正数を、減少したいときは負数を渡す。 |
| 戻り値 | 増減してできたサイズを返す。 |

■文例

```
PRINT HEX$(SETMEM(0))
PRINT HEX$(SETMEM(-&H1000))
```

<実行結果>

```
243C0
233C0
```

■ノート

▶バイト数が0のときは、増減0なのでそのときのサイズを知ることができる。

# SGN()

## 数値の符号

■機能

数値の符号を知る

■書式

**SGN(数値式)**

■引数と戻り値

| | |
|---|---|
| 引　数 | 数値式…符号を知りたい数値式を渡す。 |
| 戻り値 | 負・ゼロ・正を表わす数をそれぞれ-1、0、1を返す。 |

■文例

```
num = -127
SELECT CASE SGN(num)
CASE 1
  sign$ = "Plus"
CASE -1
  sign$ = "Minus"
CASE ELSE
  sign$ = "Zero"
END SELECT
PRINT sign$
```

<実行結果>

```
Minus
```

# SIN()

## 正弦の計算

■機能

正弦(サイン)を求める

■書式

**SIN(数値式)**

■引数と戻り値

| | |
|---|---|
| 引 数 | 数値式…ラジアンを渡す。 |
| 戻り値 | ラジアンの正弦を返す。 |

■文例

```
CONST PI = 3.14593
angle = 45
PRINT SIN(angle * PI / 180)
```

<実行結果>

```
.7078731
```

■ノート

▶参照関数：COS()、TAN()

# SPACE$()

## 空白文字列

■機能

半角の空白文字列を作る

■書式

**SPACE$(長さ)**

■引数と戻り値

| | |
|---|---|
| 引　数 | 長さ…作りたい空白文字列の長さを渡す。 |
| 戻り値 | 指定した長さの空白文字列が返される。 |

■文例

```
ymd$ = "1989" + SPACE$(3) + "06" + SPACE$(3) + "15'
PRINT ymd$
```

<実行結果>

```
1989   06   15
```

■ノート

▶長さは、0〜32767まで指定できる。

▶長さが0のときはヌルが返される。

# SPC()

## スキップ

■**機能**

PRINT および LPRINT ステートメントにおいて、表示データまたは印字データの間の間隔を空ける。

■**書式**

**SPC(文字数)**

■**引数と戻り値**

| | |
|---|---|
| 引　数 | 文字数…空けたい間隔文字数を渡す。 |
| 戻り値 | なし |

■**文例**

```
PRINT "Quick"; SPC(0); "BASIC"; SPC(1); "4.2"
```

**<実行結果>**

```
QuickBASIC 4.2
```

■**ノート**

▶コマンドの関数は、PRINT または LPRINT ステートメントの中でしか使えない。式の中では使えない。

▶文字数は、0〜32767まで指定できる。

▶文字数が 0 のときの間隔は空かない。

▶参照関数：SPACE$()、TAB()

# SQR()

## 平方根の計算

■機能

平方根を求める

■書式

**SQR(数値式)**

■引数と戻り値

| | |
|---|---|
| 引　数 | 数値式…平方根を求めたい値または数値式を渡す。 |
| 戻り値 | 数値式の平方根を返す。 |

■文例

```
org = 64
PRINT SQR(org)
```

<実行結果>

```
8
```

■ノート

▶数値式は、0以上でなければならない。

# STR$()

## 文字列への変換

**■機能**

数値を文字列に変換する

**■書式**

**STR$(数値式)**

**■引数と戻り値**

| | |
|---|---|
| 引　数 | 数値式…文字列に変えた数字を渡す。 |
| 戻り値 | 文字列を返す。 |

**■文例**

```
year = 1989: month = 6: day = 18
today$ = STR$(year) + " 年" + STR$(month) + " 月" + STR$(day) + " 日"
PRINT today$
```

**<実行結果>**

```
1989 年 6 月 18 日
```

**■ノート**

▶返される文字の先頭は、符号用になっている。引数が負ならばマイナス記号が付き、正ならば空白文字が付く。

▶参照関数：VAL()

# STRINGS$()

## 反復文字列

■機能

任意の文字から成る文字列を作る

■書式

**STRING$(文字数, {コード | 文字式})**

■引数と戻り値

| | |
|---|---|
| 引　数 | 文字数…作りたい文字列の長さを渡す。<br>コード…ASCII コード( 0 ～255)または漢字連続コード(256～11535)を数値で渡す。<br>文字式…反復のもとの文字を渡す。 |
| 戻り値 | コードに対応する文字または文字式の先頭文字を使って文字数分だけの長さの反復文字列を返す。 |

■文例

```
PRINT STRING$(60, "-")
```

<実行結果>

```
------------------------------------------------------------
```

■ノート

▶文字数は、半角・全角に関係なくどちらも1文字として数える。

▶「漢字連続コード」とは、シフトJIS漢字コード表の2バイト文字である&H8140以降を、ASCIIコードに続くものとして256〜11535までのコードを連続的に付けたコードをいう。たとえば、つぎのようになっている。

| 全角 | 連続コード |
|---|---|
| 、 | 257 |
| 。 | 258 |
| ， | 259 |
| | ⋮ |

▶文字式を渡すと、半角・全角に関係なく先頭の1文字を使って反復文字を作る。

STRING$()

# TAB()

## タブの設定

■機能

スクリーン表示およびプリンター印字の左端からのタブを設定する

■書式

**TAB(文字数)**

■引数と戻り値

| | |
|---|---|
| 引　数 | 文字数…左端から数えた位置を渡す。 |
| 戻り値 | なし |

■文例

```
PRINT "Quick"; TAB(7); "BASIC"; TAB(14); "4.2"
```

<実行結果>

```
Quick BASIC  4.2
```

■ノート

▶コマンドの関数は、PRINT または LPRINT ステートメントの中でしか使えない。式の中では使えない。

▶参照関数：SPACE$()、SPC()

# TAN()

## 正接の計算

■機能

正接(タンジェント)を求める

■書式

**TAN(数値式)**

■引数と戻り値

| | |
|---|---|
| 引　数 | 数値式…ラジアンを渡す。 |
| 戻り値 | ラジアンの正接を返す。 |

■文例

```
CONST PI = 3.14593
angle = 30
PRINT TAN(angle * PI / 180)
```

<実行結果>

```
.5783145
```

■ノート

▶参照関数：SIN()、COS()、ATN()

# TIMES$

## 現在の時刻

**■機能**

MS-DOS 上の現在の時刻を知る

**■書式**

**TIME$**

**■引数と戻り値**

| | |
|---|---|
| 引　数 | なし |
| 戻り値 | 現在の24時間制時刻をつぎの形式の文字列で返す。<br>HH：MM：SS |

**■文例**

```
PRINT TIME$, TIMER
```

**<実行結果>**

```
20:55:56        75356
```

**■ノート**

▶ MS-DOS の時刻を設定するには、TIME$ステートメントを使う。

▶参照関数：DATE$、TIMER

# TIMER

## 経過秒数

■機能

時刻を"00：00：00"からの経過秒数で知る

■書式

**TIMER**

■引数と戻り値

| | |
|---|---|
| 引　数 | なし |
| 戻り値 | 現在の時刻を深夜0時からの秒数(数字)を返す。 |

■文例

```
PRINT TIME$, TIMER
```

<実行結果>

```
20:55:56        75356
```

■ノート

▶午前7時0分3秒("07：00：03")は、25230を返す。

▶参照関数：TIME$、DATE$

TIMER

# UBOUND()

## 添字の上限

### ■機能

ある配列における添字の上限を知る

### ■書式

**UBOUND(配列 [, 次元])**

### ■引数と戻り値

| | |
|---|---|
| 引　数 | 配列名…上限を知りたい配列の名前を渡す。<br>次元　…上限を知りたい配列の次元を渡す。 |
| 戻り値 | 指定した配列の次元の上限を返す。 |

### ■文例

```
DIM linear(50), sales(1980 TO 1999, 1 TO 12)
PRINT UBOUND(linear); UBOUND(sales); UBOUND(sales, 2)
```

<実行結果>

```
50  1999  12
```

### ■ノート

▶次元を省略すると、1次元を指定したことになる。

▶参照ステートメント：DIM

▶参照関数：LBOUND()

# UCASE$()

## 大文字への変換

**■機能**

文字列におけるアルファベットの小文字部分を大文字に変換する

**■書式**

**UCASE$(文字式)**

**■引数と戻り値**

| | |
|---|---|
| 引　数 | 文字式…大文字に変換したい小文字を含む文字式を渡す。 |
| 戻り値 | 小文字の部分を大文字にした文字列を返す。 |

**■文例**

```
PRINT UCASE$("QuickＢａｓｉｃ4.2")
```

**<実行結果>**

```
QUICKＢＡＳＩＣ4.2
```

**■ノート**

▶アルファベットの小文字として、

半角のa～z

全角のａ～ｚ

のどちらでもよい。

▶参照関数：LCASE$()

UCASE$()

# VAL()

## 数値への変換

**■機能**

文字列を数値に変換する

**■書式**

**VAL(文字式)**

**■引数と戻り値**

| | |
|---|---|
| 引　数 | 文字式…数値にしたい文字列を渡す。 |
| 戻り値 | 文字列形式になっている数字を10進数にして返す。 |

**■文例**

```
PRINT JIS$("、"), CHR$(257), CHR$(VAL("&J2122"))
```

**<実行結果>**

```
2122
```

■ノート

▶数字に変換できないときは、0を返す。

▶引数の先頭に接頭記号があるつぎのタイプとして判断する。

| 接頭記号 | タイプ |
|---|---|
| &H | 16進数文字 |
| &O | 8進数文字 |
| &J | JISコード(漢字連続コードを返す) |
| &K | 句点コード(漢字連続コードを返す) |

# VARPTR() /VARPTR$()

## セグメント内での変数のオフセット・アドレス

**■機能**

現在のセグメント内での変数のオフセット・アドレスを知る。

**■書式**

**VARPTR(変数名)**
**VARPTR$(変数名)**

**■引数と戻り値**

| | |
|---|---|
| 引　数 | 変数名…オフセット・アドレスを知りたい変数の名前を渡す。 |
| 戻り値 | 引数として渡した変数のオフセット・アドレスを整数または文字列で返す。 |

**■文例**

```
FOR r = 0 TO 360 STEP 30
  DRAW "TA=" + VARPTR$(r) + "BM300,200 R100 D100 L100 U100"
NEXT
```

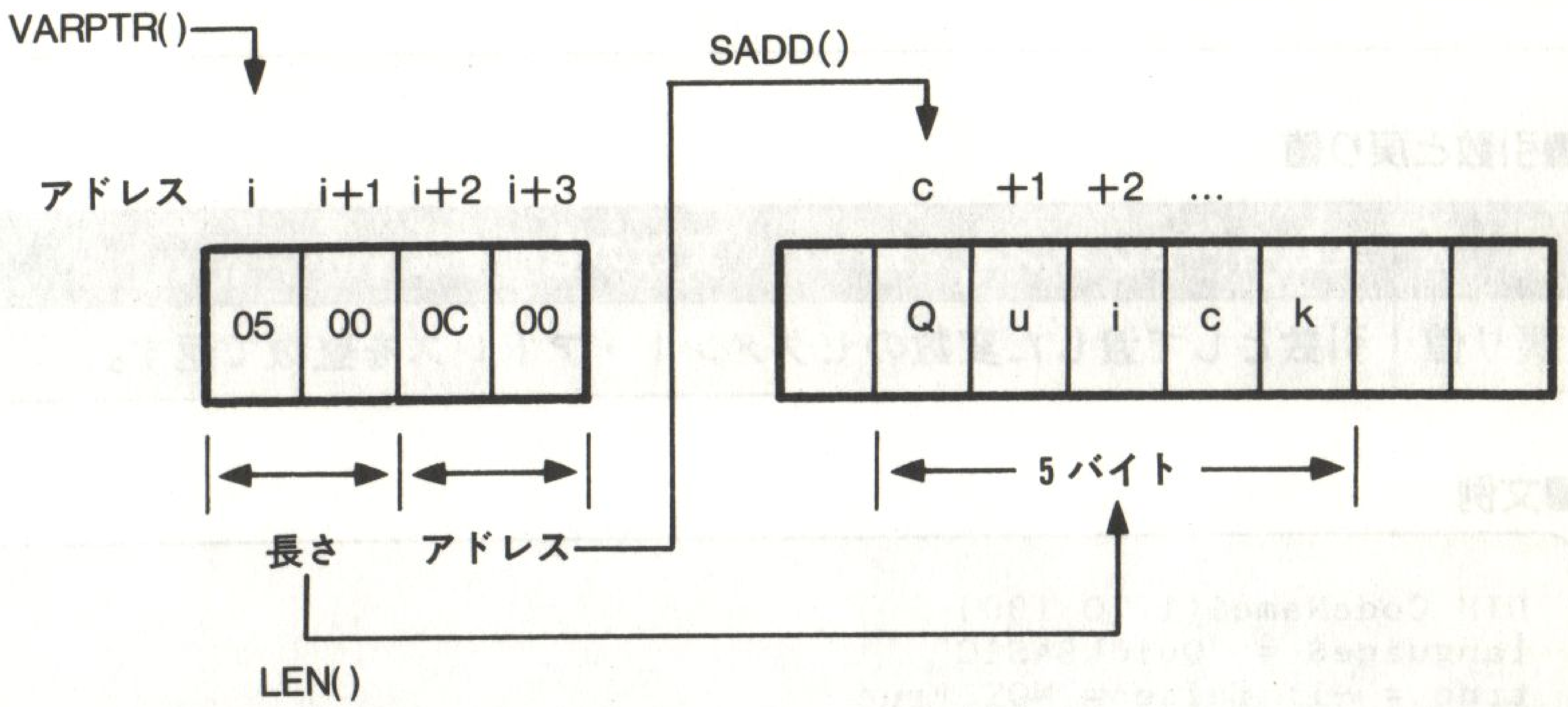
■ノート

▶ VARPTR()は整数で返し、VARPTR$()は文字列で返す。

▶文字列変数のオフセット・アドレスは、その文字列データのアドレスではなくて、4バイトの"文字列記述子"の先頭アドレスである。

文字列記述子は、最初の2バイトはその文字列の長さを管理し、残り2バイトは、文字列データが記憶されている実際のアドレスを管理している。

最初の2バイトの内容を返すのがLEN関数であり、残り2バイトの内容を返すのがSADD関数である。




▶参照ステートメント：DEF SEG、DRAW

▶参照関数：VARPTR()、VARPTR$()、SADD()

# VARSEG

## 変数のセグメント・アドレス

### ■機能

変数のセグメント・アドレスを知る

### ■書式

**VARSEG(変数名)**

### ■引数と戻り値

| | |
|---|---|
| 引　数 | 変数名…セグメントを知りたい変数の名前を渡す。 |
| 戻り値 | 引数として渡した変数のセグメント・アドレスを整数で返す。 |

### ■文例

```
DIM CodeName$(1 TO 100)
language$ = "QuickBASIC"
true = -1: false = NOT true
PRINT HEX$(VARSEG(CodeName$)), HEX$(VARPTR(CodeName$))
PRINT HEX$(VARSEG(language$)), HEX$(VARPTR(language$))
PRINT HEX$(VARSEG(true)), HEX$(VARPTR(true))
```

**<実行結果>**

```
7787          440E
7787          43F0
7787          43FA
```

VARSEG()

### ■ノート

▶参照ステートメント：DEF　SEG

▶参照関数：VARPTR()、VARPTR$()、SADD()

# [第5章]
# メタ・コマンド

この章では、QuickBASICが標準として提供している
メタ・コマンドのすべてをアルファベット順に説明します。

5

メタ・コマンドとは、プログラムの実行上、ソース・ファイルや変数の取り扱い方についてQuick BASICに指示するための命令です。全部で3つのメタ・コマンドがあります。

▶$DYNAMIC
▶$INCLUDE:
▶$STATIC

メタ・コマンドに関して、つぎのような一般的注意事項があります。

**●メタ・コマンドの名前は、必ず"$"で始まらなければならない。**
**●メタ・コマンドは、ステートメントや関数と異なり、注釈文の中に書かなければならない。**
**●1つの注釈文の中に複数のメタ・コマンドを書くことができる。それらのメタ・コマンドの間には、空白またはタブを入れなければならない。**

# この章の読み方

この章では、メタ・コマンドを以下のスタイルで記述します。

**■メタ・コマンド名**

機能についての要約が記述されています。

**■機能**

そのメタ・コマンドの働きや用途についての要約が記述されています。

**■書式**

メタ・コマンドについての完全な構文(シンタックス)が記述されています。

◉書式中の記号の記述と意味

| 用　　途 | 文字・記号 | 例 |
|---|---|---|
| そのまま書くべき箇所 | 大英文字および<br>,　”　－　;　/ | $DYNAMIC |
| 適当なものに置き換えて書くべき箇所 | 日本語の部分 | インクルード・ファイル |
| どれか1つを選ぶべき箇所 | {　\|　} | {REM \| '} |

**■引数**

引数の機能や意味について記述されています。

**■文例**

サンプルとしてその使用例を掲げてあります。

**■ノート**

補足的な説明や使用するときの注意点について記述されています。

# $DYNAMIC

## 配列の記憶領域のプログラム実行時割り当て

### ■機能

プログラムの実行中に、配列のための記憶領域をそのつど割り当てる

### ■書式

**{REM | ' }　$DYNAMIC**

### ■引数

なし

### ■文例

```
'$DYNAMIC
DIM FileEntry(999)
```

### ■ノート

▶$DYNAMIC 指定しなくても、特に指示しないかぎり、つぎのものはこの種の配列になる。

※COMMON で宣言した配列　COMMON Year(1980 TO 1989)

※添字が変数になっている配列　max = 999
DIM FileEntry(max)

▶参照ステートメント：ERASE

# $INCLUDE：

## インクルード・ファイルの取り込み

■機能

統合環境下でプログラムを実行するとき、またはコンパイルするときに別のファイルとして存在するインクルード・ファイルをソース・ファイルの一部として、指定箇所に取り込む

■書式

**{REM | ' }　$INCLUDE：　'インクルード・ファイル'**

■引数

| 引数 | 機能 |
|---|---|
| **インクルード・ファイル** | **取り込みたいインクルード・ファイルを渡す。** |

■文例

```
$INCLUDE: 'BIN¥QLB.BI'
```

■ノート

▶インクルード・ファイル名は、拡張子まで指定しなければならない。

▶インクルード・ファイル名は、'で囲まなければならない。

▶インクルード・ファイル名は、大文字でも小文字でもよい。

▶インクルード・ファイルについては、「**Vol. 1の5・7　インクルード・ファイルの管理と編集**」を参照してください。

# $STATIC

## 配列の記憶領域のコンパイル時割り当て

■機能

プログラムをコンパイルするときに、配列のための記憶領域を割り当てる

■書式

{REM | ' }　$STATIC

■引数

なし

■文例

```
'$STATIC
max = 999
DIM FileEntry(max)
```

■ノート

▶$STATIC を指定しなくても、特に指示しないかぎり、つぎのものはこの種の配列になる。

| | |
|---|---|
| ※範囲指定が明確な配列 | DIM Year(1980 TO 1989) |
| ※添字が確定している配列 | DIM code(50) |
| | CONST max = 999 |
| | DIM FileEntry(max) |

▶参照ステートメント：ERASE

# ［付録］

❶ASCIIコード
❷キー・スキャン・コード
❸エラーメッセージ
❹機能別ステートメント・関数索引
❺Ver.4.5で追加されたステートメントおよび関数

# 付録1

## ASCIIコード

| 文字 | 10進 | 8進 | 16進 | 文字 | 10進 | 8進 | 16進 |
|---|---|---|---|---|---|---|---|
| NUL | 0 | 0 | 0 | SPACE | 32 | 40 | 20 |
| SOH | 1 | 1 | 1 | ! | 33 | 41 | 21 |
| STX | 2 | 2 | 2 | " | 34 | 42 | 22 |
| ETX | 3 | 3 | 3 | # | 35 | 43 | 23 |
| EOT | 4 | 4 | 4 | $ | 36 | 44 | 24 |
| ENQ | 5 | 5 | 5 | % | 37 | 45 | 25 |
| ACK | 6 | 6 | 6 | & | 38 | 46 | 26 |
| BEL | 7 | 7 | 7 | ' | 39 | 47 | 27 |
| BS | 8 | 10 | 8 | ( | 40 | 50 | 28 |
| HT | 9 | 11 | 9 | ) | 41 | 51 | 29 |
| LF | 10 | 12 | A | * | 42 | 52 | 2A |
| HM | 11 | 13 | B | + | 43 | 53 | 2B |
| FF | 12 | 14 | C | , | 44 | 54 | 2C |
| CR | 13 | 15 | D | - | 45 | 55 | 2D |
| SO | 14 | 16 | E | . | 46 | 56 | 2E |
| SI | 15 | 17 | F | / | 47 | 57 | 2F |
| DLE | 16 | 20 | 10 | 0 | 48 | 60 | 30 |
| DC1 | 17 | 21 | 11 | 1 | 49 | 61 | 31 |
| DC2 | 18 | 22 | 12 | 2 | 50 | 62 | 32 |
| DC3 | 19 | 23 | 13 | 3 | 51 | 63 | 33 |
| DC4 | 20 | 24 | 14 | 4 | 52 | 64 | 34 |
| NAK | 21 | 25 | 15 | 5 | 53 | 65 | 35 |
| SYN | 22 | 26 | 16 | 6 | 54 | 66 | 36 |
| ETB | 23 | 27 | 17 | 7 | 55 | 67 | 37 |

| 文字 | 10進 | 8進 | 16進 | 文字 | 10進 | 8進 | 16進 |
|---|---|---|---|---|---|---|---|
| CAN | 24 | 30 | 18 | 8 | 56 | 70 | 38 |
| EM | 25 | 31 | 19 | 9 | 57 | 71 | 39 |
| SUB | 26 | 32 | 1A | : | 58 | 72 | 3A |
| ESC | 27 | 33 | 1B | ; | 59 | 73 | 3B |
| → | 28 | 34 | 1C | < | 60 | 74 | 3C |
| ← | 29 | 35 | 1D | = | 61 | 75 | 3D |
| ↑ | 30 | 36 | 1E | > | 62 | 76 | 3E |
| ↓ | 31 | 37 | 1F | ? | 63 | 77 | 3F |
| @ | 64 | 100 | 40 |  | 96 | 140 | 60 |
| A | 65 | 101 | 41 | a | 97 | 141 | 61 |
| B | 66 | 102 | 42 | b | 98 | 142 | 62 |
| C | 67 | 103 | 43 | c | 99 | 143 | 63 |
| D | 68 | 104 | 44 | d | 100 | 144 | 64 |
| E | 69 | 105 | 45 | e | 101 | 145 | 65 |
| F | 70 | 106 | 46 | f | 102 | 146 | 66 |
| G | 71 | 107 | 47 | g | 103 | 147 | 67 |
| H | 72 | 110 | 48 | h | 104 | 150 | 68 |
| I | 73 | 111 | 49 | i | 105 | 151 | 69 |
| J | 74 | 112 | 4A | j | 106 | 152 | 6A |
| K | 75 | 113 | 4B | k | 107 | 153 | 6B |
| L | 76 | 114 | 4C | l | 108 | 154 | 6C |
| M | 77 | 115 | 4D | m | 109 | 155 | 6D |
| N | 78 | 116 | 4E | n | 110 | 156 | 6E |
| O | 79 | 117 | 4F | o | 111 | 157 | 6F |
| P | 80 | 120 | 50 | p | 112 | 160 | 70 |
| Q | 81 | 121 | 51 | q | 113 | 161 | 71 |
| R | 82 | 122 | 52 | r | 114 | 162 | 72 |
| S | 83 | 123 | 53 | s | 115 | 163 | 73 |
| T | 84 | 124 | 54 | t | 116 | 164 | 74 |
| U | 85 | 125 | 55 | u | 117 | 165 | 75 |
| V | 86 | 126 | 56 | v | 118 | 166 | 76 |
| W | 87 | 127 | 57 | w | 119 | 167 | 77 |
| X | 88 | 130 | 58 | x | 120 | 170 | 78 |
| Y | 89 | 131 | 59 | y | 121 | 171 | 79 |
| Z | 90 | 132 | 5A | z | 122 | 172 | 7A |

| 文字 | 10進 | 8進 | 16進 | 文字 | 10進 | 8進 | 16進 |
|---|---|---|---|---|---|---|---|
| [ | 91 | 133 | 5B | { | 123 | 173 | 7B |
| ¥ | 92 | 134 | 5C | | 124 | 174 | 7C |
| ] | 93 | 135 | 5D | } | 125 | 175 | 7D |
| ^ | 94 | 136 | 5E | | 126 | 176 | 7E |
| _ | 95 | 137 | 5F | | 127 | 177 | 7F |
| – | 128 | 200 | 80 | | 160 | 240 | A0 |
| | 129 | 201 | 81 | 。 | 161 | 241 | A1 |
| | 130 | 202 | 82 | 「 | 162 | 242 | A2 |
| | 131 | 203 | 83 | 」 | 163 | 243 | A3 |
| | 132 | 204 | 84 | 、 | 164 | 244 | A4 |
| | 133 | 205 | 85 | ・ | 165 | 245 | A5 |
| | 134 | 206 | 86 | ヲ | 166 | 246 | A6 |
| | 135 | 207 | 87 | ァ | 167 | 247 | A7 |
| | 136 | 210 | 88 | ィ | 168 | 250 | A8 |
| | 137 | 211 | 89 | ゥ | 169 | 251 | A9 |
| | 138 | 212 | 8A | ェ | 170 | 252 | AA |
| | 139 | 213 | 8B | ォ | 171 | 253 | AB |
| | 140 | 214 | 8C | ャ | 172 | 254 | AC |
| | 141 | 215 | 8D | ュ | 173 | 255 | AD |
| | 142 | 216 | 8E | ョ | 174 | 256 | AE |
| | 143 | 217 | 8F | ッ | 175 | 257 | AF |
| | 144 | 220 | 90 | ー | 176 | 260 | B0 |
| | 145 | 221 | 91 | ア | 177 | 261 | B1 |
| | 146 | 222 | 92 | イ | 178 | 262 | B2 |
| | 147 | 223 | 93 | ウ | 179 | 263 | B3 |
| | 148 | 224 | 94 | エ | 180 | 264 | B4 |
| | 149 | 225 | 95 | オ | 181 | 265 | B5 |
| | 150 | 226 | 96 | カ | 182 | 266 | B6 |
| | 151 | 227 | 97 | キ | 183 | 267 | B7 |
| | 152 | 230 | 98 | ク | 184 | 270 | B8 |
| | 153 | 231 | 99 | ケ | 185 | 271 | B9 |
| | 154 | 232 | 9A | コ | 186 | 272 | BA |
| | 155 | 233 | 9B | サ | 187 | 273 | BB |
| | 156 | 234 | 9C | シ | 188 | 274 | BC |
| | 157 | 235 | 9D | ス | 189 | 275 | BD |

| 文字 | 10進 | 8進 | 16進 | 文字 | 10進 | 8進 | 16進 |
|---|---|---|---|---|---|---|---|
| | 158 | 236 | 9E | セ | 190 | 276 | BE |
| | 159 | 237 | 9F | ソ | 191 | 277 | BF |
| タ | 192 | 300 | C0 | | 224 | 340 | E0 |
| チ | 193 | 301 | C1 | | 225 | 341 | E1 |
| ツ | 194 | 302 | C2 | | 226 | 342 | E2 |
| テ | 195 | 303 | C3 | | 227 | 343 | E3 |
| ト | 196 | 304 | C4 | | 228 | 344 | E4 |
| ナ | 197 | 305 | C5 | | 229 | 345 | E5 |
| ニ | 198 | 306 | C6 | | 230 | 346 | E6 |
| ヌ | 199 | 307 | C7 | | 231 | 347 | E7 |
| ネ | 200 | 310 | C8 | | 232 | 350 | E8 |
| ノ | 201 | 311 | C9 | | 233 | 351 | E9 |
| ハ | 202 | 312 | CA | | 234 | 352 | EA |
| ヒ | 203 | 313 | CB | | 235 | 353 | EB |
| フ | 204 | 314 | CC | | 236 | 354 | EC |
| ヘ | 205 | 315 | CD | | 237 | 355 | ED |
| ホ | 206 | 316 | CE | | 238 | 356 | EE |
| マ | 207 | 317 | CF | | 239 | 357 | EF |
| ミ | 208 | 320 | D0 | | 240 | 360 | F0 |
| ム | 209 | 321 | D1 | | 241 | 361 | F1 |
| メ | 210 | 322 | D2 | | 242 | 362 | F2 |
| モ | 211 | 323 | D3 | | 243 | 363 | F3 |
| ヤ | 212 | 324 | D4 | | 244 | 364 | F4 |
| ユ | 213 | 325 | D5 | | 245 | 365 | F5 |
| ヨ | 214 | 326 | D6 | | 246 | 366 | F6 |
| ラ | 215 | 327 | D7 | | 247 | 367 | F7 |
| リ | 216 | 330 | D8 | | 248 | 370 | F8 |
| ル | 217 | 331 | D9 | | 249 | 371 | F9 |
| レ | 218 | 332 | DA | | 250 | 372 | FA |
| ロ | 219 | 333 | DB | | 251 | 373 | FB |
| ワ | 220 | 334 | DC | | 252 | 374 | FC |
| ン | 221 | 335 | DD | | 253 | 375 | FD |
| ゛ | 222 | 336 | DE | | 254 | 376 | FE |
| ゜ | 223 | 337 | DF | | 255 | 377 | FF |

# ［付録2］

## キー・スキャン・コード



この表の見方

▶表は、キー・スキャン・コード順になっています。

▶"キー"は、実際のキーボード・トップのことです。

▶"スキャン・コード"は、KEY(n)ステートメントで使うキー・コードです。

▶"ASCII"の欄は、そのキーだけを押したときに、INKEY$()が返す値です。

▶"SHIFT＋"の欄は、SHIFT キーを押しながらそれぞれのキーを押したときに、INKEY$()が返す値です。

▶"CTRL＋"の欄は、CTRL キーを押しながらそれぞれのキーを押したときに、INKEY$()が返す値です。

▶"カナ＋"の欄は、カナキーを押しながらそれぞれのキーを押したときに、INKEY$()が返す値です。

▶"SHIFT＋カナ＋"の欄は、SHIFT キーと カナ キーを押しながらそれぞれのキーを押したときに、INKEY$()が返す値です。

▶表の中の＾が前に付いている文字は、制御コード文字です。

▶表の中の"00"が前に付いている10進数および16進数は、第1バイトがヌルで第2バイトがそのコードです。

| キー | スキャン・コード | | ASCII | | | SHIFT + | | | CTRL + | | | カナ + | | | カナ + SHIFT + | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 10進 | 16進 | 文字 | 10進 | 16進 | 文字 | 10進 | 16進 | 文字 | 10進 | 16進 | 文字 | 10進 | 16進 | 文字 | 10進 | 16進 |
| <ESC> | 0 | 0 | ^[ | 27 | 1B | ^[ | 27 | 1B | ^[ | 27 | 1B | | | | | | |
| 1 ! ヌ | 1 | 1 | 1 | 49 | 31 | ! | 33 | 21 | | | | ヌ | 199 | C7 | ァ | 167 | A7 |
| 2 " フ | 2 | 2 | 2 | 50 | 32 | " | 34 | 22 | | | | フ | 204 | CC | ゥ | 169 | A9 |
| 3 # ア ァ | 3 | 3 | 3 | 51 | 33 | # | 35 | 23 | | | | ア | 177 | B1 | ェ | 170 | AA |
| 4 $ ウ ゥ | 4 | 4 | 4 | 52 | 34 | $ | 36 | 24 | | | | ウ | 179 | B3 | ォ | 171 | AB |
| 5 % エ ェ | 5 | 5 | 5 | 53 | 35 | % | 37 | 25 | | | | エ | 180 | B4 | ャ | 172 | AC |
| 6 & オ ォ | 6 | 6 | 6 | 54 | 36 | & | 38 | 26 | | | | オ | 181 | B5 | ュ | 173 | AD |
| 7 ' ヤ ャ | 7 | 7 | 7 | 55 | 37 | ' | 39 | 27 | | | | ヤ | 212 | D4 | ョ | 174 | AE |
| 8 ( ユ ュ | 8 | 8 | 8 | 56 | 38 | ( | 40 | 28 | | | | ユ | 213 | D5 | ヲ | 166 | A6 |
| 9 ) ヨ ョ | 9 | 9 | 9 | 57 | 39 | ) | 41 | 29 | | | | ヨ | 214 | D6 | | | |
| 0 0 ワ ヲ | 10 | A | 0 | 48 | 30 | 0 | 48 | 30 | | | | ワ | 220 | DC | | | |
| - = ホ | 11 | B | - | 45 | 2D | = | 61 | 3D | | | | ホ | 206 | CE | | | |
| ^ ^ ヘ | 12 | C | ^ | 94 | 5E | ^ | 94 | 5E | ^^ | 30 | 1E | ヘ | 205 | CD | | | |
| ¥ \| ー | 13 | D | ¥ | 92 | 5C | \| | 124 | 7C | ^¥ | 28 | 1C | ー | 176 | B0 | | | |
| < BS > | 14 | E | ^H | 8 | 8 | ^H | 8 | 8 | ^H | 8 | 8 | | | | | | |
| < TAB > | 15 | F | ^I | 9 | 9 | ^I | 9 | 9 | ^I | 9 | 9 | | | | | | |
| Q タ | 16 | 10 | q | 113 | 71 | Q | 81 | 51 | ^Q | 17 | 11 | タ | 192 | C0 | | | |
| W テ | 17 | 11 | w | 119 | 77 | W | 87 | 57 | ^W | 23 | 17 | テ | 195 | C3 | | | |
| E イ | 18 | 12 | e | 101 | 65 | E | 69 | 45 | ^E | 5 | 5 | イ | 178 | B2 | | | |
| R ス | 19 | 13 | r | 114 | 72 | R | 82 | 52 | ^R | 18 | 12 | ス | 189 | BD | | | |
| T カ | 20 | 14 | t | 116 | 74 | T | 84 | 54 | ^T | 20 | 14 | カ | 182 | B6 | | | |
| Y ン | 21 | 15 | y | 121 | 79 | Y | 89 | 59 | ^Y | 25 | 19 | ン | 221 | DD | | | |
| U ナ | 22 | 16 | u | 117 | 75 | U | 85 | 55 | ^U | 21 | 15 | ナ | 197 | C5 | | | |
| I ニ | 23 | 17 | i | 105 | 69 | I | 73 | 49 | ^I | 9 | 9 | ニ | 198 | C6 | | | |
| O ラ | 24 | 18 | o | 111 | 6F | O | 79 | 4F | ^O | 15 | F | ラ | 215 | D7 | | | |
| P セ | 25 | 19 | p | 112 | 70 | P | 80 | 50 | ^P | 16 | 10 | セ | 190 | BE | | | |
| @ ゛ | 26 | 1A | @ | 64 | 40 | | 126 | 7E | | | | ゛ | 222 | DE | | | |
| [ { ゜ 「 | 27 | 1B | [ | 91 | 5B | { | 123 | 7B | ^[ | 27 | 1B | ゜ | 223 | DF | 「 | 162 | A2 |
| <RETURN> | 28 | 1C | ^M | 13 | D | ^M | 13 | D | ^M | 13 | D | | | | | | |
| A チ | 29 | 1D | a | 97 | 61 | A | 65 | 41 | ^A | 1 | 1 | チ | 193 | C1 | | | |
| S ト | 30 | 1E | s | 115 | 73 | S | 83 | 53 | ^S | 19 | 13 | ト | 196 | C4 | | | |
| D シ | 31 | 1F | d | 100 | 64 | D | 68 | 44 | ^D | 4 | 4 | シ | 188 | BC | | | |
| F ハ | 32 | 20 | f | 102 | 66 | F | 70 | 46 | ^F | 6 | 6 | ハ | 202 | CA | | | |
| G キ | 33 | 21 | g | 103 | 67 | G | 71 | 47 | ^G | 7 | 7 | キ | 183 | B7 | | | |
| H ク | 34 | 22 | h | 104 | 68 | H | 72 | 48 | ^H | 8 | 8 | ク | 184 | B8 | | | |
| J マ | 35 | 23 | j | 106 | 6A | J | 74 | 4A | ^J | 10 | A | マ | 207 | CF | | | |
| K ノ | 36 | 24 | k | 107 | 6B | K | 75 | 4B | ^K | 11 | B | ノ | 201 | C9 | | | |
| L リ | 37 | 25 | l | 108 | 6C | L | 76 | 4C | ^L | 12 | C | リ | 216 | D8 | | | |

| キー | スキャン・コード | | ASCII | | | SHIFT + | | | CTRL + | | | カナ + | | | カナ + SHIFT + | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 10進 | 16進 | 文字 | 10進 | 16進 | 文字 | 10進 | 16進 | 文字 | 10進 | 16進 | 文字 | 10進 | 16進 | 文字 | 10進 | 16進 |
| ; + レ | 38 | 26 | ; | 59 | 3B | + | 43 | 2B | | | | レ | 218 | DA | | | |
| : * ケ | 39 | 27 | : | 58 | 3A | * | 42 | 2A | | | | ケ | 185 | B9 | | | |
| ] } ム 」 | 40 | 28 | ] | 93 | 5D | } | 125 | 7D | ^] | 29 | 1D | ム | 209 | D1 | 」 | 163 | A3 |
| Z ツ | 41 | 29 | z | 122 | 7A | Z | 90 | 5A | ^Z | 26 | 1A | ツ | 194 | C2 | | | |
| X サ | 42 | 2A | x | 120 | 78 | X | 88 | 58 | ^X | 24 | 18 | サ | 187 | BB | | | |
| C ソ | 43 | 2B | c | 99 | 63 | C | 67 | 43 | ^C | 3 | 3 | ソ | 191 | BF | | | |
| V ヒ | 44 | 2C | v | 118 | 76 | V | 86 | 56 | ^V | 22 | 16 | ヒ | 203 | CB | | | |
| B コ | 45 | 2D | b | 98 | 62 | B | 66 | 42 | ^B | 2 | 2 | コ | 186 | BA | | | |
| N ミ | 46 | 2E | n | 110 | 6E | N | 78 | 4E | ^N | 14 | E | ミ | 208 | D0 | | | |
| M モ | 47 | 2F | m | 109 | 6D | M | 77 | 4D | ^M | 13 | D | モ | 211 | D3 | | | |
| , < ネ 、 | 48 | 30 | , | 44 | 2C | < | 60 | 3C | | | | ネ | 200 | C8 | 、 | 164 | A4 |
| . > ル ° | 49 | 31 | . | 46 | 2E | > | 62 | 3E | | | | ル | 217 | D9 | ° | 161 | A1 |
| / ? メ ・ | 50 | 32 | / | 47 | 2F | ? | 63 | 3F | | | | メ | 210 | D2 | ・ | 165 | A5 |
| ロ | 51 | 33 | | | | | 95 | 5F | ^ | 31 | 1F | ロ | 219 | DB | | | |
| <SPACE> | 52 | 34 | | 32 | 20 | | 32 | 20 | | 32 | 20 | | | | | | |
| < XFER > | 53 | 35 | | | | | | | | | | | | | | | |
| <ROLLUP> | 54 | 36 | | 0081 | 0051 | | 0081 | 0051 | | | | | | | | | |
| <ROLLDW> | 55 | 37 | | 0073 | 0049 | | 0073 | 0049 | | | | | | | | | |
| < INS > | 56 | 38 | | 0082 | 0052 | | 0082 | 0052 | | | | | | | | | |
| < DEL > | 57 | 39 | | 0083 | 0053 | | 0083 | 0053 | | | | | | | | | |
| < ↑ > | 58 | 3A | | 0072 | 0048 | | 0072 | 0048 | | | | | | | | | |
| < ← > | 59 | 3B | | 0075 | 004B | | 0075 | 004B | | | | | | | | | |
| < → > | 60 | 3C | | 0077 | 004D | | 0077 | 004D | | | | | | | | | |
| < ↓ > | 61 | 3D | | 0080 | 0050 | | 0080 | 0050 | | | | | | | | | |
| < CLR > | 62 | 3E | | 0071 | 0047 | | 0071 | 0047 | | | | | | | | | |
| < HELP > | 63 | 3F | | 0079 | 004F | | 0079 | 004F | | | | | | | | | |
| - | 64 | 40 | - | 45 | 2D | - | 45 | 2D | | | | | | | | | |
| / | 65 | 41 | / | 47 | 2F | / | 47 | 2F | | | | | | | | | |
| 7 | 66 | 42 | 7 | 55 | 37 | 7 | 55 | 37 | | | | | | | | | |
| 8 | 67 | 43 | 8 | 56 | 38 | 8 | 56 | 38 | | | | | | | | | |
| 9 | 68 | 44 | 9 | 57 | 39 | 9 | 57 | 39 | | | | | | | | | |
| * | 69 | 45 | * | 42 | 2A | * | 42 | 2A | | | | | | | | | |
| 4 | 70 | 46 | 4 | 52 | 34 | 4 | 52 | 34 | | | | | | | | | |
| 5 | 71 | 47 | 5 | 53 | 35 | 5 | 53 | 35 | | | | | | | | | |
| 6 | 72 | 48 | 6 | 54 | 36 | 6 | 54 | 36 | | | | | | | | | |
| + | 73 | 49 | + | 43 | 2B | + | 43 | 2B | | | | | | | | | |
| 1 | 74 | 4A | 1 | 49 | 31 | 1 | 49 | 31 | | | | | | | | | |
| 2 | 75 | 4B | 2 | 50 | 32 | 2 | 50 | 32 | | | | | | | | | |

| キー | スキャン・コード | | ASCII | | | SHIFT + | | | CTRL + | | | カナ + | | | カナ + SHIFT + | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 10進 | 16進 | 文字 | 10進 | 16進 | 文字 | 10進 | 16進 | 文字 | 10進 | 16進 | 文字 | 10進 | 16進 | 文字 | 10進 | 16進 |
| 3 | 76 | 4C | 3 | 51 | 33 | 3 | 51 | 33 | | | | | | | | | |
| = | 77 | 4D | = | 61 | 3D | = | 61 | 3D | | | | | | | | | |
| 0 | 78 | 4E | 0 | 48 | 30 | 0 | 48 | 30 | | | | | | | | | |
| , | 79 | 4F | , | 44 | 2C | , | 44 | 2C | | | | | | | | | |
| . | 80 | 50 | . | 46 | 2E | . | 46 | 2E | | | | | | | | | |
| < NFER > | 81 | 51 | | | | | | | | | | | | | | | |
| < STOP > | 96 | 60 | | | | | | | | | | | | | | | |
| < COPY > | 97 | 61 | | | | | | | | | | | | | | | |
| f ・ 1 | 98 | 62 | | 0059 | 003B | | 0084 | 0054 | | | | | | | | | |
| f ・ 2 | 99 | 63 | | 0060 | 003C | | 0085 | 0055 | | | | | | | | | |
| f ・ 3 | 100 | 64 | | 0061 | 003D | | 0086 | 0056 | | | | | | | | | |
| f ・ 4 | 101 | 65 | | 0062 | 003E | | 0087 | 0057 | | | | | | | | | |
| f ・ 5 | 102 | 66 | | 0063 | 003F | | 0088 | 0058 | | | | | | | | | |
| f ・ 6 | 103 | 67 | | 0064 | 0040 | | 0089 | 0059 | | | | | | | | | |
| f ・ 7 | 104 | 68 | | 0065 | 0041 | | 0090 | 005A | | | | | | | | | |
| f ・ 8 | 105 | 69 | | 0066 | 0042 | | 0091 | 005B | | | | | | | | | |
| f ・ 9 | 106 | 6A | | 0067 | 0043 | | 0092 | 005C | | | | | | | | | |
| f ・ 10 | 107 | 6B | | 0068 | 0044 | | 0093 | 005D | | | | | | | | | |
| < SHIFT > | 112 | 70 | | | | | | | | | | | | | | | |
| < CAPS > | 113 | 71 | | | | | | | | | | | | | | | |
| < カナ > | 114 | 72 | | | | | | | | | | | | | | | |
| < GRPH > | 115 | 73 | | | | | | | | | | | | | | | |
| < CTRL > | 116 | 74 | | | | | | | | | | | | | | | |

# 付録3

## エラー・メッセージ

対応するFORがありません(ERR =1)
構文に誤りがあります(ERR =2)
対応するGOSUBがありません(ERR =3)
READで読み込むDATAがありません(ERR =4)
引数が許される範囲ではありません(ERR =5)
計算結果がオーバーフローしました(ERR =6)
メモリーが足りません(ERR =7)
ラベルが定義されていません(ERR =8)
配列の添字が許される範囲にありません(ERR =9)
宣言の重複は許されません(ERR =10)
0で除算しました(ERR =11)
ダイレクト・モードでは許されないステートメントです(ERR =12)
変数,式の型が一致しません(ERR =13)
文字領域が足りません(ERR =14)
文字式が複雑すぎます(ERR =16)
継続できません(ERR =17)
FUNCTIONが定義されていません(ERR =18)
RESUMEがありません(ERR =19)
エラー未発生でRESUMEが実行されました(ERR =20)
デバイスから応答がありません(ERR =24)
デバイスが異常です(ERR =25)
対応するNEXTがありません(ERR =26)
用紙がありません(ERR =27)
対応するWENDがありません(ERR =29)
対応するWHILEがありません(ERR =30)
ラベルが重複しています(ERR =33)
SUBが定義されていません(ERR =35)

引数の数があいません(ERR =37)
配列が宣言されていません(ERR =38)
CASE ELSE が必要です(ERR =39)
引数として変数が必要です(ERR =40)
指定した FIELD 長を超えています(ERR =50)
内部エラーが発生しました(ERR =51)
ファイル名・ファイル番号が不適当です(ERR =52)
ファイルが見つかりません(ERR =53)
ファイルモードが不適当です(ERR =54)
既に OPEN されています(ERR =55)
FIELD で指定された変数は利用できません(ERR =56)
I/O デバイスに異常があります(ERR =57)
既にファイルが存在します(ERR =58)
レコード長が不適当です(ERR =59)
ディスクに書き込むスペースがありません(ERR =61)
ファイルの終わりに達しました(ERR =62)
レコード番号が不適当です(ERR =63)
ファイル名が不適当です(ERR =64)
ファイルが多すぎます(ERR =67)
デバイスが準備されていません(ERR =68)
通信用バッファがオーバーフローしました(ERR =69)
ファイル・ディスクが書き込み禁止です(ERR =70)
ディスクが準備されていません(ERR =71)
ディスクが異常です(ERR =72)
現在サポートされていません(ERR =73)
ディスク間での名前の変更はできません(ERR =74)
パス名が不適当です(ERR =75)
パス名が見つかりません(ERR =76)
未定義のエラーが発生しました(ERR =??)

# ［付録 4］

## 機能別ステートメント・関数索引

| ステートメント関数 | 機能 | ページ |
| --- | --- | --- |

### ■ファイル

■スクリーン

■キーボード



■プリンター



■通信



■時間

## ■定数と変数

## ■メモリー管理



## ■プログラム制御

## ■ DOS

## ■文字列処理

■算術と数値処理

## ■IO 制御



## ■エラー処理



## ■サウンド

# [付録5]

## Ver.4.5で追加されたステートメントおよび関数

QuickBASIC Ver.4.5では、次の6つのステートメントと1つの関数が追加されました。

- ON UEVENT GOSUB
- PLAY
- PLAY {ON | OFF | STOP}
- SLEEP
- SOUND
- UEVENT
- PLAY()

以下、これらのステートメントおよび関数の機能について解説します。

# ON UEVENT GOSUB

## ユーザー定義のイベント・トラップ用サブ・ルーチンへの分岐

■機能

UEVENTステートメントで定義したユーザー定義のイベント・トラップ用サブ・ルーチンへ分岐する

■書式

**ON UEVENT GOSUB {行番号|行ラベル}**

■引数

| 引数 | 機能 |
|---|---|
| 行番号<br>行ラベル | 分岐先を行番号または行ラベルで指定する。 |

■文例

```
UEVENT ON
ON UEVENT GOSUB myEventTrap
DO
  .
  .
LOOP

myEventTrap:
UEVENT OFF
  .
  .
  .
```

■ノート

▶参照ステートメント：UEVENT

# PLAY

## 音楽の演奏

■機能

文字列で指定した音楽を演奏する

■書式

**PLAY　音楽文字列**

■引数

| 引数 | 機能 | | |
|---|---|---|---|
| 音楽文字列 | 音楽を演奏するためのマクロコマンドから成る文字列を指定する。 | | |
| | オクターブ | > | 1オクターブ上げる。 |
| | | < | 1オクターブ下げる。 |
| | | On | n(0～6)オクターブに設定する。 |
| | トーン | C | ド |
| | | D | レ |
| | | E | ミ |
| | | F | ファ |
| | | G | ソ |
| | | A | ラ |
| | | B | シ |
| | | # | CからBの直後に付けると半音上がる。 |
| | | + | CからBの直後に付けると半音上がる。 |
| | | － | CからBの直後に付けると半音下がる。 |

| | | |
|---|---|---|
| トーン | Nn | 絶対音符 n( 0 または 1 〜84)を鳴らす。<br>1〜84は、12音符(C,C #, D,D #, E,F,F #, G,G #, A,A #, B)、7 オクターブに順次対応している。<br>0 で休符になる。 |
| 長さ | Ln | 1 音符の長さ n を全音符から64分音符で指定する( 1 〜64)。 |
| | MN | ノーマル（1 音符の7/8の長さで) |
| | ML | レガート（1 音符の長さで) |
| | MS | スタッカート（1 音符の3/4の長さで) |
| テンポ | Tn | 1 分間における 4 分音符の数(32〜255、デフォルトは120) |
| | Pn | 停止の長さ n を全音符から64分音符で指定する( 1 〜64)。 |
| 演奏 | MF | フォアグランドで演奏する(すでに行なわれている演奏が終わるまで、次の演奏を開始しない)。 |
| | MB | バックグランドで演奏する(バックで演奏しながらプログラムを実行できる)。 |
| | ”X”＋VARPTR$(文字列変数) | 音楽文字列として文字列変数を使う。 |

■文例

```
CLS
PRINT "さあ、みんなでドレミの唄を歌いましょう！"
PRINT "イチ・ニー・サン、ハイ！"
Lemon$ = "L3D L7E L8F L8F L8E D L2F P16"

PLAY "L3C L8D E P8 C P8 E P8 C P8 L2E P16":  ' ドはドーナッツのド
PLAY "X" + VARPTR$(Lemon$):                  ' レはレモンのレ
  .
  .
```

# PLAY

## 演奏イベント・トラッピングの設定

■機能

演奏イベント・トラッピングを有効または無効にする

■書式

**PLAY ｛ON | OFF | STOP｝**

■引数

| 引　数 | 機　能 |
|---|---|
| ON | 演奏イベント・トラッピングを有効にする。<br>有効であるあいだ、ON PLAY(n) GOSUB ステートメントで設定した最小音符数未満になったかどうかチェックし、未満になったときはそのイベント・トラッピングのサブルーチンを実行する。 |
| OFF<br>STOP | 演奏イベント・トラッピングを無効にする。<br>OFF は、そのイベントを記憶しないが、STOP は記憶しているので、トラッピングできるようになると ON PLAY ルーチンを実行する。 |

■文例

```
CLS
ON PLAY(2) GOSUB aSongOfDOREMI
PLAY ON
PRINT "さあ、みんなでドレミの唄を歌いましょう！"
PRINT "イチ・ニー・サン"
PLAY "L2C D E"
END

aSongOfDOREMI:
  Lemon$ = "L3D L7E L8F L8F L8E D L2F P16"
  PLAY "L3C L8D E P8 C P8 E P8 C P8 L2E P16"
  PLAY "L3D L7E L8F L8F L8E D L2F P16"
RETURN
```

■ノート

▶参照ステートメント：ON event GOSUB

# SLEEP

## プログラムの実行の一時停止

### ■機能

指定した時間が経過するまで、いずれかのキーが押されるまで、あるいはイベント・トラッピングが行なわれるまでプログラムの実行を停止する

### ■書式

**SLEEP ［秒数］**

### ■引数

| 引 数 | 機 能 |
|---|---|
| 秒数 | 停止させたい時間を秒数で指定する。<br>0としたときまたは省略したときは、キーが押されるまであるいはイベント・トラッピングがあるまでプログラムの実行を停止する。 |

### ■文例

```
LOCATE 23, 10
PRINT "プリンタのスイッチを確認してください..."
SLEEP 10
    .
    .
    .
```

# SOUND

## 音を出す

■機能

スピーカーから音を出す

■書式

**SOUND 周波数 , 長さ**

■引数

| 引　数 | 機　　能 |
| --- | --- |
| 周波数 | 出したい音の周波数(37〜32767)を整数で指定する。 |
| 長さ | 1秒／18.2を単位とする音の長さを指定する。 |

■文例

```
oneSecond = 18.2
SOUND 900, oneSecond / 2 ────── 900ヘルツを0.5秒出す
```

# UEVENT

## ユーザー定義のイベント・トラッピングの設定

■機能

ユーザー定義のイベント・トラッピングを有効または無効にする

■書式

**UEVENT ｛ON ｜ OFF ｜ STOP｝**

■引数

| 引数 | 機能 |
|---|---|
| ON | ユーザー定義のイベント・トラッピングを有効にする。<br>有効であるあいだ、ON uevent(n) GOSUB ステートメントで設定したユーザーイベントが発生したかどうかチェックし、発生したときはそのイベント・トラッピングのサブルーチンを実行する。 |
| OFF<br>STOP | ユーザー定義のイベント・トラッピングを無効にする。<br>OFF は、そのイベントを記憶しないが、STOP は記憶しているので、トラッピングできるようになると ON UEVENT ルーチンを実行する。 |

■文例

```
UEVENT ON
ON UEVENT GOSUB myEventTrap
DO
  .
  .
LOOP

myEventTrap:
UEVENT OFF
  .
  .
  .
```

■ノート

▶参照ステートメント：ON UEVENT GOSUB

# PLAY()

## 音符の数

■機能

BGM (バックグラウンドミュージック) のキュー (Queue) における音符の数を知る

■書式

**PLAY（ダミー）**

■引数と戻り値

| | |
|---|---|
| 引　数 | ダミー … 数値(何でもよい)を渡す。 |
| 戻り値 | BGMにおける残りの音符数を返す。<br>フォアグラウンドのときは、0を返す。 |

■文例

```
PLAY(1)
```

■ノート

▶フォアグランドおよびバックグランドについては、PLAYステートメントを参照。

# 索　引

## ●D



## ●E



## ●F



## ●G



## ●H



## ●I

## ◉さ行



## ◉た行

## ◉な行



## ◉は行

## ◉ま行



## ◉や行



## ◉ら行

**著者略歴**

**阿部友計**（あべ・ともかず）
昭和48年、日本大学商学部卒業
昭和55年、成蹊大学大学院経営学 研究科博士課程修了。
**〈著書〉**
「dBASEⅢ PLUSハンディマニュアル」、「dBASEⅢ PLUS
プログラミング技法」、「MS-DOS Ver. 3.3ハンドブック」
（以上、共著、ナツメ社）。

**本書に対するご質問**

本書に対するご質問、ご意見がありましたら、下記の
住所までお手紙にてご連絡下さい。電話によるお問い
合せはご遠慮下さい。
ご質問はできるだけ、往復はがきか返信用封筒同封の
うえいただければ幸いです。

ナツメ出版企画株式会社　書籍第2編集部
質問係

**QuickBASICハンドブック Vol.2リファレンス**

**著　者**　**阿部友計**

**発行者**　**田村正隆**

**制　作**　**ナツメ出版企画株式会社**
東京都千代田区神田神保町1-52加州ビル3F（〒101）
電話　03（3295）3921（代表）
**発行所**　**株式会社ナツメ社**
東京都千代田区神田神保町1-52加州ビル2F（〒101）
電話　03（3291）1257（代表）
振替　東京 3-58661
**印刷所**　**ラン印刷社**

ISBN4-8163-0961-6　Printed in Japan

ハンディ・リファレンス……38

# QuickBASIC ハンドブックvol.2［リファレンス］

発行日――1991年2月20日
著者――阿部友計
発行者――田村正隆
制作――ナツメ出版企画株式会社
郵便番号101
東京都千代田区神田神保町1-52加州ビル3F
電話　03(3295)3921
発行所――株式会社ナツメ社
郵便番号101
東京都千代田区神田神保町1-52加州ビル2F
電話　03(3291)1257
振替　東京3-58661
定価――1,300円
〈落丁・乱丁はお取り替えします〉



handy reference

このハンドブックは、QuickBASICのすべての
ステートメント、関数の使い方を文例つきでまとめました。
プログラム開発を強力に支援してくれる統合環境については
「QuickBASICハンドブック Vol.1 基本操作」にまとめてあります。
このハンドブック2冊を利用しながら、
快適で効率のよいQuickBASICのプログラミングを行なってください。




ナツメ社｜定価1,300円［本体1,262円］
ISBN4-8163-0961-6 C2054 P1300E